Informatica

Informatica® Cloud Data Integration

REST V2 Connector

Informatica Cloud Data Integration REST V2 Connector
February 2025

Publication Date: 2025-02-10

Table of Contents

Preface

Use *REST V2 Connector* to learn how to read from or write to a web service that supports REST API by using Data Integration. Learn to create a REST V2 connection, develop and run mappings and mapping tasks in Data Integration.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Intelligent Cloud Services web site

You can access the Informatica Intelligent Cloud Services web site at http://www.informatica.com/cloud. This site contains information about Informatica Cloud integration services.

Informatica Intelligent Cloud Services Communities

Use the Informatica Intelligent Cloud Services Community to discuss and resolve technical issues. You can also find technical tips, documentation updates, and answers to frequently asked questions.

Access the Informatica Intelligent Cloud Services Community at:

https://network.informatica.com/community/informatica-network/products/cloud-integration

Developers can learn more and share tips at the Cloud Developer community:

https://network.informatica.com/community/informatica-network/products/cloud-integration/cloud-developers

Informatica Intelligent Cloud Services Marketplace

Visit the Informatica Marketplace to try and buy Data Integration Connectors, templates, and mapplets:

https://marketplace.informatica.com/

Data Integration connector documentation

You can access documentation for Data Integration Connectors at the Documentation Portal. To explore the Documentation Portal, visit https://docs.informatica.com.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Intelligent Cloud Services Trust Center

The Informatica Intelligent Cloud Services Trust Center provides information about Informatica security policies and real-time system availability.

You can access the trust center at https://www.informatica.com/trust-center.html.

Subscribe to the Informatica Intelligent Cloud Services Trust Center to receive upgrade, maintenance, and incident notifications. The Informatica Intelligent Cloud Services Status page displays the production status of all the Informatica cloud products. All maintenance updates are posted to this page, and during an outage, it will have the most current information. To ensure you are notified of updates and outages, you can subscribe to receive updates for a single component or all Informatica Intelligent Cloud Services components. Subscribing to all components is the best way to be certain you never miss an update.

To subscribe, on the Informatica Intelligent Cloud Services Status page, click **SUBSCRIBE TO UPDATES**. You can choose to receive notifications sent as emails, SMS text messages, webhooks, RSS feeds, or any combination of the four.

Informatica Global Customer Support

You can contact a Global Support Center through the Informatica Network or by telephone.

To find online support resources on the Informatica Network, click **Contact Support** in the Informatica Intelligent Cloud Services Help menu to go to the **Cloud Support** page. The **Cloud Support** page includes system status information and community discussions. Log in to Informatica Network and click **Need Help** to find additional resources and to contact Informatica Global Customer Support through email.

The telephone numbers for Informatica Global Customer Support are available from the Informatica web site at https://www.informatica.com/services-and-training/support-services/contact-us.html.

CHAPTER 1

Introduction to REST V2 Connector

Use REST V2 Connector to interact with web service applications built on REST architecture. You can use REST V2 Connector in a Source transformation, Target transformation, or midstream in a Web Services transformation.

REST V2 Connector supports Swagger specification 2.0 and OpenAPI 3.0.x that helps to interact with web service applications. The Swagger or OpenAPI specification file must be in JSON format.

You can use REST V2 Connector midstream in a mapping to pass a single or multiple requests to a web service application and process the response data. You can also pass data obtained from multiple transformations in the mapping pipeline and process the data.

When you use REST V2 Connector midstream in a mapping, you first create a business service for the operation that you want to perform in the web service application. You then associate the business service in a Web Services transformation midstream in a mapping to read from or write data to the web service application. For example, you can use REST V2 Connector as a midstream transformation to perform PUT operation on a web service application.

You can switch mappings to advanced mode to include transformations and functions that enable advanced functionality.

REST V2 Connector assets

Create assets in Data Integration to integrate data using Rest V2 Connector.

You can use Source, Target, and Midstream transformation types in the following Data Integration assets:

- Mapping
- Mapping task

When you run a mapping or mapping task, the session log is saved to the following directory: `<Secure Agent installation directory>/apps/Data_Integration_Server/logs`.

For more information about configuring assets and transformations, see *Mappings*, *Transformations*, and *Tasks* in the Data Integration documentation.

Basic structure of the Swagger and OpenAPI specification file

Swagger is a specification for documenting REST API. It specifies the format to describe REST web services.

REST V2 Connector supports swagger specification 2.0 and OpenAPI 3.0.x to interact with web service applications. When you configure a Rest V2 connection, you can specify the path of the Swagger or OpenAPI file in the **Swagger File Path** property.

The swagger or OpenAPI specification file contains the following details:

- metadata of the API
- server details
- operations
- path parameters
- query parameters
- header fields
- request details
- response details

Servers

You can specify an array of server URL in the servers section in the OpenAPI specification.

The following example shows how servers are defined in the OpenAPI specification file:

```
{
  "servers": [
    {
      "url": "https://development.gigantic-server.com/v1",
      "description": "Development server"
    },
    {
      "url": "https://staging.gigantic-server.com/v1",
      "description": "Staging server"
    },
    {
      "url": "https://api.gigantic-server.com/v1",
      "description": "Production server"
    }
  ]
}
```

You can define servers globally at the path level or for specific operations. However, Rest V2 Connector honors the first server listed in the specification file to connect to the REST endpoint.

Alternatively, if you want to specify the server URL when you create a Rest V2 connection, you can add the host URL in the **Advanced Fields** property in the following format: `hosturl:schemes://host`

For example, `hosturl:https://openapiworld.com`

The server specified in the connection properties takes precedence over the server specified in the specification file.

For more information on creating a Rest V2 connection, see Rest V2 connection properties.

Note: You can't specify variables in a server URL.

In the swagger specification, you can define a server URL using the host keyword. You cannot define multiple servers in the swagger specification file.

The following example shows how servers are defined in the swagger specification file:

```
{
  "host": "petstore.swagger.io",
  "basePath": "/v2",
        "schemes": [
        "https"
]
}
```

Parameters

In the OpenAPI specification, you can define parameters at the path level and for specific operations. The parameters at the path level are applicable to all operations defined under the path. The parameters at the operation level are applicable only to the operations under which it is defined.

If the same parameters are defined at the path level and operation level, the operation level parameters take precedence.

You can define the following parameter types:

- path parameters
- query parameters
- header parameters
- cookie parameters

The following example shows how the parameters are defined in the OpenAPI specification file:

```
"paths": {
"/pet": {
"put": {
    "tags": [
    "pet"
],
"summary": "Update an existing pet",
    "description": "Update an existing pet by Id",
    "operationId": "updatePet",
    "parameters": [
{
"name": "status",
"in": "query",
"description": "Status values that need to be considered for filter",
"required": false,
"explode": true,
"schema": {
        "type": "string",
        "default": "available",
        }
    }
]
```

Note: Ensure that the parameters section in the OpenAPI specification contains the schema type.

In the swagger specification file, you can define parameters for each operation.

The following example shows how parameters are defined in the swagger specification file:

```
"parameters": [
{
 "in": "body",
 "name": "user",
 "description": "The user to create.",
 "schema": {
  "$ref": "#/definitions/User"
   }
        }
]
```

Request body

Request body defines the structure of the endpoint request body.

In the OpenAPI specification file, you can define parameters and requestBody to connect to a REST endpoint.

The following example shows how the requestBody is defined in the OpenAPI specification file:

```
"operationId": "addPet",
"requestBody": {
 "description": "Create a new pet in the store",
 "content": {
  "application/json": {
   "schema": {
   "$ref": "#/components/schemas/Pet"
    }
   },
"application/xml": {
 "schema": {
 "$ref": "#/components/schemas/Pet"
   }
  }
 }
}
```

In the swagger specification file, you can define the parameters section to send a request to the REST endpoint.

The following example shows how the request body is defined in the swagger specification file:

```
{
"paths": {
    "/users": {
            "post": {
            "summary": "Creates a new user.",
            "consumes": [
            "application/json"
],
            "parameters": [
            {
            "in": "body",
            "name": "user",
            "description": "The user to create.",
            "schema": {
                "$ref": "#/definitions/User"
                    }
        }
]
```

Response body

Response body defines the response fields of the endpoint. Rest V2 Connector supports only the successful response with HTTP status code 200. For all responses other than status code 200, the field mapping displays a static field named Response_Port.

The following example shows the response body section in the OpenAPI specification file:

```
"responses": {
"200": {
 "description": "Successful operation",
 "content": {
 "application/xml": {
  "schema": {
   "$ref": "#/components/schemas/Pet"
     }
    },
 "application/json": {
  "schema": {
    "$ref": "#/components/schemas/Pet"
        }
    }
```

```
        }
      }
    }
```

The following example shows the response body section in the swagger specification file:

```
    "responses": {
    "200": {
        "description": "A User object",
        "schema": {
        "$ref": "#/definitions/User"
            }
        }
    }
```

Media types and operations

Media types

You can define media types in request and response definitions when you connect to the REST endpoint.

In Swagger, you can specify the following media types in request and response definitions to process data:

- application/xml
- application/json
- application/x-www-form-urlencoded
- JSON subtype. For example: `application/`**`hal+`**`json`
- JSON custom type. For example: `application/`**`vnd.ds.abc.v1+`**`json`
- Extended JSON mime type. For example: `application/vnd.ds.abc.v1+json;`**`version=q1`**
- text/xml

If a swagger definition contains multiple mime types in consumes and produces attribute values, the REST V2 Connector considers the payload to be of first mime type in the list.

In OpenAPI, you can specify the following media types in request and response definitions to process data:

- application/xml
- application/json

Consider the following guidelines when you define media types in OpenAPI:

- The default request and response media type for all operations is `application/json`.
- If `application/json` media type is not available for an operation, the task considers `application/xml` as the request and response media type.
- If `application/json` or `application/xml` media type is not available for an operation, the request message template displays a static field named Request_Port and the field mapping displays a static field named Response_Port.

Operations

You can define operations to connect to the REST endpoint.

You can use the following REST methods or operations in source, target, and midstream transformations in Swagger or OpenAPI:

- GET

- PUT
- POST
- DELETE
- OPTIONS
- HEAD
- PATCH

Array of objects

You can define an array of objects for `application/json` media type in the OpenAPI specification file in the request and response definitions.

Inline array in the request and response body

The requestBody or response where all the fields of the request body are defined in the `requestBody` section.

For example,

```
"/users_inline": {
"post": {
  "description": "Add Multiple Users",
  "operationId": "users_inline",
  "requestBody": {
  "description": "Add Multiple Users",
  "content": {
  "application/json": {
  "schema": {
   "type": "array",
   "items": {
   "type": "object",
   "properties": {
      "first_name": {
      "type": "string"
       },
       "last_name": {
        "type": "string"
       }
      }
     }
    }
   }
  }
}
"responses": {
  "200": {
   "description": "add User Response",
    "content": {
     "application/json": {
       "schema": {
       "type": "array",
           "items": {
           "type": "object",
        "properties": {
          "first_name": {
          "type": "string"
           },
          "last_name": {
          "type": "string"
           },
          "id": {
               "type": "string"
               }
             }
            }
           }
          }
        }
       }
    }
```

Inline ref array in the request and response body

The requestBody or response where the schema type is array and it contains the reference object `$ref`.

For example,

```
    "post": {
    "description": "Add Multiple Users",
    "operationId": "users_inline_ref",
    "requestBody": {
      "description": "Add new Users",
      "content": {
       "application/json": {
        "schema": {
        "type": "array",
        "items": {
        "$ref": "#/components/schemas/user"
         }
        }
       }
      }
     }
    "responses": {
      "200": {
       "description": "add User Response",
        "content": {
        "application/json": {
        "schema": {
         "type": "array",
         "items": {
          "$ref": "#/components/schemas/user"
            }
        }
       }
      }
     }
    }
```

Array in component in the request and response body

The requestBody or response where schema contains only the reference object `$ref` pointing to a component.

For example,

```
    "post": {
    "description": "Add Multiple Users",
     "operationId": "users",
     "requestBody": {
       "description": "Add new Users",
       "content": {
       "application/json": {
       "schema": {
       "$ref": "#/components/schemas/user_list_inline"
        }
       }
      }
     }
    "components": {
      "schemas": {
```

```
  "user_list": {
  "type": "array",
  "items": {
  "$ref": "#/components/schemas/user"
   }
  }
 }
}
"responses": {
 "200": {
  "description": "add User Response",
  "content": {
  "application/json": {
  "schema": {
  "$ref": "#/components/schemas/user_list"
     }
    }
   }
  }
}
"components": {
  "schemas": {
   "user_list": {
    "type": "array",
     "items": {
     "$ref": "#/components/schemas/user"
          }
       }
     }
 }
```

Complex object in the request and response body

The requestBody or response that contains array fields along with simple fields.

For example,

```
"description": "Add Complex Object",
 "operationId": "aad_complex_object",
 "requestBody": {
   "description": "Add new Users",
   "content": {
   "application/json": {
    "schema": {
    "$ref": "#/components/schemas/complex_object"
               }
     }
    }
  }
"components": {
   "schemas": {
    "complex_object": {
    "type": "object",
     "properties": {
       "sample_string": {
        "type": "string"
         },
        "user_array": {
         "type": "array",
          "items": {
          "$ref": "#/components/schemas/user"
          }
         }
        }
       }
     }
    }
"responses": {
  "200": {
    "description": "add User Response",
     "content": {
     "application/json": {
       "schema": {
       "$ref": "#/components/schemas/complex_object"
         }
       }
     }
   }
 }

"components": {
  "schemas": {
   "complex_object": {
    "type": "object",
    "properties": {
     "sample_string": {
      "type": "string"
          },
      "user_array": {
       "type": "array",
       "items": {
       "$ref": "#/components/schemas/user"
       }
      }
     }
    }
  }
}
```

XML objects in swagger specification

You can configure XML objects such as XML attributes, the wrapper object, and the namespace object in the swagger specification file.

You can define attributes only for XML objects. You cannot define attributes for XML elements. You can use wrapper objects for a simple type array element only when the array element has inline definitions. Use qualified and unqualified namespace objects in response for source, midstream, and target transformations and in request for midstream and target transformations.

An XML attribute is always of type string. Even if the swagger definition has attribute with type as number, REST V2 Connector always treats the attribute as string. The XSD generated contains the data type as string for the XML attribute.

Note: Ensure that the Swagger definition does not contain multiple namespaces.

Swagger definition for an XML attribute:

```
"XMLAttrArray_Request##body##Employee##dep##Mangr" : {

"properties" : {

"desg" : {

"type" : "string",

"xml": { "attribute": true }

}, "mid" : { "type" : "string" },
```

XML attribute sample:

```
<dep>

<Mangr desg="Director">
```

```
<mid>em09</mid>

<mname>mg</mname>

</Mangr>

<id>did</id>

<name>dname</name>

</dep>
```

Note: If the REST endpoint returns a response in an application/xml format and the XML element in the response contains both an attribute and an element value, the mapping does not process the response. For example, you cannot process an XML sample `<emp desg="Director">John</emp>`, where the **emp** element contains both an attribute **desg** and an element value **John**.

Swagger definition for a Wrapper object:

```
"books" : {

"type" : "array",

"items" : { "type" : "string" },

"xml": {

"wrapped" : "true", "name": "books-array"

}

}
```

Wrapper object sample:

```
<books-array>

<books>1</books>

<books>2</books>

</books-array>
```

Swagger definition for a Namespace object:

```
root" : {

"properties" : {

"table" : {

"$ref" : "#/definitions/NSResReq_Request##body##root##table"

}

},

"xml":{

"prefix": "h",

"namespace": "http://www.w3.org/TR/html4/"

}

}
```

Namespace object sample:

```
<h:root xmlns:h="http://www.w3.org/TR/html4/">
```

```
<table>

<tr>

<td>Apples</td>

<td>Bananas</td>

</tr>

</table>

</h:root>
```

CHAPTER 2

Connections for REST V2

Create a REST V2 connection to make calls to a web service application.

When you create a connection, specify the swagger specification file and the authentication method. You can specify the following authentication methods:

- Standard
- OAuth 2.0 client credentials
- OAuth 2.0 authorization code
- JWT bearer token
- API key

You can use OAuth 2.0 client credentials or OAuth 2.0 authorization code authentication to connect to the authorization server in a Virtual Private Cloud network for the authorization process.

If your REST endpoint does not have a swagger specification, you can generate the swagger specification file from Administrator. Click **Administrator** > **Swagger Files** to generate a swagger specification file.

For information about the parameters that you need to define in a Swagger file, see "Generating a Swagger File" on page 62.

Prerequisites

Before you configure a REST V2 connection, be sure to complete the prerequisites.

- Install the Secure Agent on a 64-bit machine.
- Ensure that the machine hosting the Secure Agent has a minimum memory size of 2048 MB.

Connect to REST V2

Let's configure the REST V2 connection properties to interact with web service applications built on REST architecture.

Before you begin

Before you get started, be sure to complete the prerequisites.

Check out “Prerequisites” on page 18 to learn more about the required prerequisites.

Connection details

The following table describes the basic connection properties:

Property	Description
Connection Name	Name of the connection. Each connection name must be unique within the organization. Connection names can contain alphanumeric characters, spaces, and the following special characters: _ . + -, Maximum length is 255 characters.
Description	Description of the connection. Maximum length is 4000 characters.
Type	REST V2
Runtime Environment	Name of the runtime environment where you want to run tasks. Select a Secure Agent, Hosted Agent, or serverless runtime environment.

Authentication types

You can configure standard, OAuth 2.0 client credentials, OAuth 2.0 authorization code, JWT bearer token, and API key authentication types to connect to a REST endpoint.

Select the required authentication method and then configure the authentication-specific parameters.

Standard authentication

Standard authentication requires an authentication user ID and password to connect to a REST endpoint. When you configure a standard authentication type, you can further configure basic and OAuth authentication types.

Note: Digest authentication is not applicable.

The following table describes the basic connection properties for standard authentication:

Property	Description
Authentication Type	The authentication type that you can use when you select the Standard authentication. You can select one of the following authentication types: - BASIC - OAUTH - NONE Default is NONE.
Auth User ID	The user name to log in to the web service application when you select the standard authentication. Required for Basic authentication type.
Auth Password	The password associated with the user name when you select the standard authentication. Required for Basic authentication type.

Property	Description
OAuth Consumer Key	The client key associated with the web service application. Required only for OAuth authentication type.
OAuth Consumer Secret	The client password to connect to the web service application. Required only for OAuth authentication type.
OAuth Token	The access token to connect to the web service application. Required only for OAuth authentication type.
OAuth Token Secret	The password associated with the OAuth token. Required only for OAuth authentication type.
Swagger File Path	The path of the Swagger file or OpenAPI file. You can specify one of the following file paths: - Path and file name of the Swagger or OpenAPI file on the Secure Agent machine. - The URL on which the Swagger or OpenAPI file is hosted. The hosted URL must return the content of the file without prompting for further authentication and redirection. For example, the path of the Swagger file can be: `C:\Swagger\sampleSwagger.json` The user must have the read permission for the folder and the file.

OAuth 2.0 client credentials authentication

OAuth 2.0 client credentials authentication requires at a minimum the client ID, access token URL, client secret, scope, and the access token.

The following table describes the basic connection properties for OAuth 2.0 client credentials authentication:

Property	Description
Access Token URL	Access token URL configured in your application.
Client ID	Client ID of your application.
Client Secret	Client secret of your application.
Scope	Specifies access control if the API endpoint has defined custom scopes. Enter scope attributes, each separated by a space. For example: `root_readonly root_readwrite manage_app_users`
Access Token Parameters	Additional parameters to use with the access token URL. Define the parameters in the JSON format. For example, `[{"Name":"resource","Value":"https://<serverName>"}]`

Property	Description
Client Authentication	Select an option to send the client ID and client secret for authorization either in the request body or in the request header. Default is **Send Client Credentials in Body**.
Generate Access Token	Generates an access token based on the information provided in the above fields.
Access Token	The access token value. Enter the access token value or click **Generate Access Token** to populate the access token value. To pass the generate access token call through a proxy server, you must configure an unauthenticated proxy server in the Secure Agent properties. The proxy configured in the connection configuration does not apply to the generate access token call.
Swagger File Path	The path of the Swagger file or OpenAPI file. You can specify one of the following file paths: - Path and file name of the Swagger or OpenAPI file on the Secure Agent machine. - The URL on which the Swagger or OpenAPI file is hosted. The hosted URL must return the content of the file without prompting for further authentication and redirection. For example, the path of the swagger file can be: `C:\swagger\sampleSwagger.json` The user must have the read permission for the folder and the file.

OAuth 2.0 authorization code authentication

To use authorization code authentication, register the following Informatica redirect URL in your application:

```
https://<Informatica cloud hosting facility for your organization>/ma/proxy/oauthcallback
```

If the access token expires and the error codes `400`, `401`, and `403` are returned in the response, Informatica redirect URL, which is outside the customer firewall, tries to connect to the endpoint and retrieves a new access token.

The following table describes the basic connection properties for OAuth 2.0 authorization code authentication:

Property	Description
Authorization Token URL	Authorization server URL configured in your application.
Access Token URL	Access token URL configured in your application.
Client ID	Client ID of your application.
Client Secret	Client secret of your application.

Property	Description
Scope	Specifies access control if the API endpoint has defined custom scopes. Enter scope attributes, each separated by a space. For example, `root_readonly root_readwrite manage_app_users`
Access Token Parameters	Additional parameters to use with the access token URL. Define the parameters in the JSON format. For example, `[{"Name":"resource","Value":"https://<serverName>"}]`
Authorization Code Parameters	Additional parameters to use with the authorization token URL. Define the parameters in the JSON format. For example, `[{"Name":"max_age","Value":60},{"Name":"state","Value":"test"}]`
Client Authentication	Select an option to send the client ID and client secret for authorization either in the request body or in the request header. Default is **Send Client Credentials in Body**.
Generate Access Token	Generates an access token and refreshes the token based on the information provided in the above fields.
Access Token	The access token value. Enter the access token value or click **Generate Access Token** to populate the access token value. To pass the generate access token call through a proxy server, you must configure an unauthenticated proxy server in the Secure Agent properties. The proxy configured in the connection configuration does not apply to the generate access token call.

Property	Description
Refresh Token	The refresh token value. Enter the refresh token value or click **Generate Access Token** to populate the refresh token value. If the access token is not valid or expires, the Secure Agent fetches a new access token with the help of refresh token. If the refresh token expires, you must either provide a valid refresh token or regenerate a new refresh token by clicking **Generate Access Token**.
Swagger File Path	The path of the Swagger file or OpenAPI file. You can specify one of the following file paths: - Path and file name of the Swagger or OpenAPI file on the Secure Agent machine. - The URL on which the Swagger or OpenAPI file is hosted. The hosted URL must return the content of the file without prompting for further authentication and redirection. For example, the path of the swagger file can be: `C:\swagger\sampleSwagger.json` The user must have the read permission for the folder and the file.

JWT bearer token authentication

JWT bearer token authentication requires at a minimum the JWT header, JWT payload, and authorization server URL.

The following table describes the basic connection properties for JWT bearer token authentication:

Property	Description

JWT Header	JWT header in JSON format.
	Sample:
	`{`
	`"alg":"RS256",`
	`"kid":"xxyyzz"`
	`}`
	You can configure `HS256` and `RS256` algorithms.

JWT Payload	JWT payload in JSON format.
	Sample:
	`{`
	`"iss":"abc",`
	`"sub":"678",`
	`"aud":"https://api.box.com/oauth2/token",`
	`"box_sub_type":"enterprise",`
	`"exp":"120",`
	`"jti":"3ee9364e"`
	`}`
	The expiry time represented as **exp** is the relative time in seconds. The expiry time is calculated in the UTC format from the token issuer time (`iat`).
	When `iat` is defined in the payload and the expiry time is reached, mappings and Generate Access Token fails. To generate a new access token, you must provide a valid `iat` in the payload.
	If `iat` is not defined in the payload, the expiry time is calculated from the current timestamp.
	To pass the expiry time as a string value, enclose the value with double quotes. For example:
	`"exp":"120"`
	To pass the expiry time as an integer value, do not enclose the value with double quotes.
	For example,
	`"exp":"120"`

Authorization Server	Access token URL configured in your application.
KeyStore File Path	The absolute path of the keystore file that contains the keys and certificates required to establish a two-way secure communication with the REST API. Specify a directory path that is available on each Secure Agent machine. You can also configure the keystore file name and path as a JVM option or import the certificate to any directory. For the serverless runtime environment, specify the keystore file path in the serverless agent directory. For example, `/home/cldagnt/SystemAgent/serverless/configurations/ssl_store/<cert_name>.jks`

Property	Description
KeyStore Password	The password for the keystore file required for a secure communication. You can also configure the keystore password as a JVM option.
Private Key Alias	Alias name of the private key used to sign the JWT payload.
Private Key Password	The password for the keystore file required for a secure communication. The private key password must be same as the keystore password.
Access Token	The access token value. Enter the access token value or click **Generate Access Token** to populate the access token value. To pass the generate access token call through a proxy server, you must configure an unauthenticated proxy server in the Secure Agent properties. The proxy configured in the connection configuration does not apply to the generate access token call.
Swagger File Path	The path of the Swagger file or OpenAPI file. You can specify one of the following file paths: - Path and file name of the Swagger or OpenAPI file on the Secure Agent machine. - The URL on which the Swagger or OpenAPI file is hosted. The hosted URL must return the content of the file without prompting for further authentication and redirection. For example, the path of the swagger file can be: `C:\swagger\sampleSwagger.json` The user must have the read permission for the folder and the file.

Advanced settings

The following table describes the advanced connection properties for JWT bearer token authentication:

Property	Description
Authorization Advanced Properties	Additional parameters to use with the access token URL. Parameters must be defined in the JSON format. For example, `[\{"Name":"client_id","Value":"abc"},\{"Name":"client_secret","Value":"abc"}]`
TrustStore File Path	The absolute path of the truststore file that contains the TLS certificate to establish a one-way or two-way secure connection with the REST API. Specify a directory path that is available on each Secure Agent machine. You can also configure the truststore file name and password as a JVM option or import the certificate to the following directory: `<Secure Agent installation directory\jre\lib\security\cacerts`. For the serverless runtime environment, specify the truststore file path in the serverless agent directory. For example, `/home/cldagnt/SystemAgent/serverless/configurations/ssl_store/<cert_name>.jks`
TrustStore Password	The password for the truststore file that contains the SSL certificate. You can also configure the truststore password as a JVM option.
Proxy Type	Type of proxy. Select one of the following options: - No Proxy. Bypasses the proxy server configured in the agent or the connection properties. - Platform Proxy. Considers the proxy configured in the agent. - Custom Proxy. Considers the proxy configured in the connection properties.

Property	Description
Proxy Configuration	The format required to configure proxy. You can configure proxy using the following format: `<host>:<port>` You cannot configure an authenticated proxy server.
Advanced Fields	Enter the arguments that the agent uses when connecting to a REST endpoint. When you specify multiple arguments, separate each argument by a semicolon. For example, `connectiondelaytime:10000;retryattempts:5` You can specify the following arguments: - **ConnectionTimeout**. The wait time in milliseconds to get a response from a REST endpoint. The connection ends after the connection timeout is over. Default is the timeout defined in the endpoint API. **Note:** If you define both the REST V2 connection timeout and the endpoint API timeout, the connection ends at the shortest defined timeout. - **connectiondelaytime**. The delay time in milliseconds to send a request to a REST endpoint. Default is 10000. - **retryattempts**. Number of times the connection is attempted when 400 and 500 series error codes are returned in the response. Default is 3. Specify 0 to disable the retry attempts. - **qualifiedSchema**. Determines if the schema selected is qualified or unqualified. Default is false.

API key authentication

API key authentication allows you to provide a unique key and a corresponding value to authenticate API calls made to the REST endpoint.

The following table describes the basic connection properties for API key authentication:

Property	Description
Key	The unique API key that REST V2 Connector uses to authenticate the API calls made to the REST endpoint.
Value	The value corresponding to the API key that is required to make the API calls.

Property	Description
Add API Key to	Determines if the API key and its corresponding value must be sent as a request header or a query parameter to make API calls to the REST endpoint. Select one of the following options: - Request Header - Query Parameter
Swagger File Path	The path of the Swagger file or OpenAPI file. You can specify one of the following file paths: - Path and file name of the Swagger or OpenAPI file on the Secure Agent machine. - The URL on which the Swagger or OpenAPI file is hosted. The hosted URL must return the content of the file without prompting for further authentication and redirection. For example, the path of the swagger file can be: `C:\swagger\sampleSwagger.json` The user must have the read permission for the folder and the file.

Advanced settings

The following table describes the advanced connection properties:

Property	Description
TrustStore File Path	The absolute path of the truststore file that contains the TLS certificate to establish a one-way or two-way secure connection with the REST API. Specify a directory path that is available on each Secure Agent machine. You can also configure the truststore file name and password as a JVM option or import the certificate to the following directory: `<Secure Agent installation directory\jre\lib\security\cacerts.` For the serverless runtime environment, specify the truststore file path in the serverless agent directory. For example, `/home/cldagnt/SystemAgent/serverless/configurations/ssl_store/<cert_name>.jks`
TrustStore Password	The password for the truststore file that contains the SSL certificate. You can also configure the truststore password as a JVM option.
KeyStore File Path	The absolute path of the keystore file that contains the keys and certificates required to establish a two-way secure communication with the REST API. Specify a directory path that is available on each Secure Agent machine. You can also configure the keystore file name and path as a JVM option or import the certificate to any directory. For the serverless runtime environment, specify the keystore file path in the serverless agent directory. For example, `/home/cldagnt/SystemAgent/serverless/configurations/ssl_store/<cert_name>.jks`
KeyStore Password	The password for the keystore file required for secure communication. You can also configure the keystore password as a JVM option.

Property	Description
Proxy Type	Type of proxy. Select one of the following options: - No Proxy: Bypasses the proxy server configured in the agent or the connection properties. - Platform Proxy: Considers the proxy configured in the agent. - Custom Proxy: Considers the proxy configured in the connection properties.
Proxy Configuration	The format required to configure proxy. Configure proxy using the following format: `<host>:<port>` You cannot configure an authenticated proxy server.
Advanced Fields	Enter the arguments that the agent uses when connecting to a REST endpoint. When you specify multiple arguments, separate each argument by a semicolon. For example, `connectiondelaytime:10000;retryattempts:5` You can specify the following arguments: - **ConnectionTimeout**. The wait time in milliseconds to get a response from a REST endpoint. The connection ends after the connection timeout is over. Default is the timeout defined in the endpoint API. **Note:** If you define both the REST V2 connection timeout and the endpoint API timeout, the connection ends at the shortest defined timeout. - **connectiondelaytime**. The delay time in milliseconds to send a request to a REST endpoint. Default is 10000. - **retryattempts**. Number of times the connection is attempted when 400 and 500 series error codes are returned in the response. Default is 3. Specify 0 to disable the retry attempts. - **qualifiedSchema**. Determines if the schema selected is qualified or unqualified. Default is false.

Related links

“Secure communication with TLS authentication” on page 29

“Secure communication in a serverless runtime environment” on page 32

“Swagger specification file in a serverless runtime environment” on page 32

“Rules and guidelines for a REST V2 connection” on page 34

“Rules and guidelines for runtime environment” on page 33

Secure communication with TLS authentication

Configure TLS authentication to establish one-way or two-way secure communication between the Secure Agent and the REST API over TLS.

To establish one-way secure communcation, perform the following steps:

1. Generate the truststore. For more information on the steps, see *Generate a Truststore*.

2. Configure the REST V2 connection for one-way SSL. You can specify the truststore file and truststore password in the connection, or set them in the JVM options of the Secure Agent.

To establish two-way secure communcation, you must first configure one-way secure communication, and then perform the following steps:

1. Generate the keystore. For more information on the steps, see *Generate a Keystore*.
2. Configure the REST V2 connection for two-way SSL. You can specify the keystore file and keystore password in the connection, or set them in the JVM options of the Secure Agent.

If you specify keystore and truststore properties in the connection and in the JVM options, the Secure Agent processes the certificates based on the properties configured in the connection.

Generate a truststore

To generate a truststore, you need a server certificate. Get the server certificate and perform the following steps to generate the truststore:

1. Import the server certificate to either of the following directories available within your Secure Agent installation:
   - `<Secure Agent installation directory>\jdk\jre\lib\security\cacerts`
   - `<Secure Agent installation directory>\jdk8\jre\lib\security\cacerts`
2. To generate the truststore, run the following command from the command line:
   ```
   keytool -importcert -alias <Specify alias name here> -file <Specify server certificate
   here> -keystore <Specify the name of custom truststore to be generated> -storepass
   <Specify password for the custom truststore>
   ```
   For example, `keytool -importcert -alias RESTV2CACert -file ca.pem -keystore sampletruststore -storepass JKSTrustStorePassword`

   In the example, a truststore file is generated by the name *sampletruststore* and password *JKSTrustStorePassword*.

Generate a keystore

To generate a keystore, you need a client certificate and a client private key. Get the client certificate and client private key, and then perform the following steps to generate the keystore:

1. Import the server certificate to either of the following directories available within your Secure Agent installation:
   - `<Secure Agent installation directory>\jdk\jre\lib\security\cacerts`
   - `<Secure Agent installation directory>\jdk8\jre\lib\security\cacerts`

2. To generate the keystore, run the following command from the command line:
   `openssl pkcs12 -export -in <Specify client certificate here> -inkey <Specify client private key here> -name "<Specify any name here>" -passout pass:<Specify password for the keystore to be generated> -out <Specify name for the keystore with p12 extension>`

   For example, `openssl pkcs12 -export -in /home/samplefolder/certs/client-cert.pem -inkey /home/samplefolder/certs/client-key.pem -name "restclient" -passout pass:PKCSKeyStorePassword -out samplekeystore.p12`

   In the example, a keystore file by the name *samplekeystore.p12* is generated in the PKCS12 format.

   To convert the keystore file from .p12 format to .jks format, run the following command from the command line:

   `keytool -importkeystore -srckeystore <Specify name of the p12 keystore file> -srcstoretype pkcs12 -srcstorepass <Specify password for generated p12 keystore file> -destkeystore <Specify name for the JKS keystore file> -deststoretype JKS -deststorepass <Specify password for the JKS keystore file>`

   Note: Ensure that the password specified in `-srcstorepass` must be the same as the `-deststorepass`.

   For example, `keytool -importkeystore -srckeystore samplekeystore.p12 -srcstoretype pkcs12 -srcstorepass PKCSKeyStorePassword -destkeystore keystore -deststoretype JKS -deststorepass PKCSKeyStorePassword`

   In the example, a keystore file is generated by the name *samplekeystore* and password *PKCSKeyStorePassword*.

Configuring one-way or two-way secure communication

You can configure a connection for one-way or two-way SSL.

Configuring the connection for one-way SSL

You can either specify the name of the truststore file and truststore password in the TrustStore File Name and TrustStore Password fields in the connection properties. Alterntaively, you can set the truststore file name and truststore password in the JVM options in the Secure Agent properties.

1. Click **Administrator** > **Runtime Environments**, and select an agent.
2. Select **Type** as DTM under **System Configuration Details**.
3. Add the following JVM options:
   - `JVMOption1=-Djavax.net.ssl.trustStore=<absolute path of the .jks truststore file>`
   - `JVMOption2=-Djavax.net.ssl.trustStorePassword=<truststore password>`

Configuring the connection for two-way SSL

You can either specify the name of the keystore file and keystore password in the KeyStore File Name and KeyStore Password connection properties. Alternatively, you can set the keystore file and keystore password in the JVM options in the Secure Agent properties.

To use two-way SSL, you must first configure one-way SSL, and then perform the following steps to configure two-way SSL:

1. Click **Administrator** > **Runtime Environments**, and select an agent.
2. Select **Type** as DTM under **System Configuration Details**.
3. Add the following JVM options:
   - `JVMOption3=-Djavax.net.ssl.keyStore=<absolute path of the .jks keystore file>`

- `JVMOption4=-Djavax.net.ssl.keyStorePassword=<keystore password>`

Secure communication in a serverless runtime environment

When you use the serverless runtime environment, you can configure TLS authentication and establish one-way or two-way secure communication with the REST API.

Ensure that the certificates are in the `.jks` format.

To configure a secure REST V2 connection using the serverless runtime environment, complete the following prerequisite tasks to add the TLS certificates to the serverless runtime location:

1. Create the following structure for the serverless agent configuration in AWS:
   `<Supplementary file location>/serverless_agent_config`
2. For one-way secure communication, add the truststore certificates and for the two-way secure communication, add the truststore and keystore certificates in the Amazon S3 bucket in the following location in your AWS account:
   `<Supplementary file location>/serverless_agent_config/SSL`
3. Copy the following code snippet to a text editor:
   ```
   version: 1
   agent:
     agentAutoApply:
       general:
         sslStore:
           - fileCopy:
               sourcePath: SSL/<RESTV2_trustStore_cert_name>.jks
           - fileCopy:
               sourcePath: SSL/<RESTV2_keyStore_cert_name>.jks
   ```
   where the source path is the directory of the certificate files in AWS.
4. Ensure that the syntax and indentations are valid, and then save the file as `serverlessUserAgentConfig.yml` in the following AWS location:
   `<Supplementary file location>/serverless_agent_config`

   When the .yml file runs, the SSL certificates are copied from the AWS location to the serverless agent directory.
5. In the REST V2 connection properties, specify the following certificate path in the serverless agent directory in the **TrustStore File Path** and **KeyStore File Path** fields:
   `/home/cldagnt/SystemAgent/serverless/configurations/ssl_store/<cert_name>.jks`

Swagger specification file in a serverless runtime environment

To configure a swagger file in a serverless runtime environment, be sure to complete the prerequisites.

In the serverless runtime environment, you can configure a swagger file in one of the following ways:

- Provide the swagger file public hosted URL in the **Swagger File Path** connection property. Ensure that the URL must return the content of the file without prompting for further authentication and redirection.
- Place the swagger file in the serverless agent directory.

To configure a swagger file in a serverless runtime environment, complete the following prerequisite tasks to add the swagger file to the serverless runtime location:

1. Create the following structure for the serverless agent configuration in AWS or Azure:
   `<Supplementary file location>/serverless_agent_config`
2. Add the swagger specification file in the Amazon S3 bucket or Azure container in the following location in your AWS or Azure account:
   `<Supplementary file location>/serverless_agent_config/restv2`
   a. Copy the following code snippet to a text editor:
      ```
      version: 1
      agent:
        dataIntegrationServer:
          autoApply:
            restv2:
              swaggers:
                - fileCopy:
                    sourcePath: restv2/<swagger_file_name1>.json
                - fileCopy:
                    sourcePath: restv2/<swagger_file_name2>.json
      ```
      where the source path is the directory path of the swagger files in AWS or Azure.
3. Ensure that the syntax and indentations are valid, and then save the file as `serverlessUserAgentConfig.yml` in the following AWS or Azure location:
   `<Supplementary file location>/serverless_agent_config`

   When the .yml file runs, the SSL certificates are copied from the AWS or Azure location to the serverless agent directory.
4. In the REST V2 connection properties, specify the following swagger path in the serverless agent directory in the **Swagger File Path** field:
   `/home/cldagnt/SystemAgent/serverless/configurations/restv2/<swagger_file_name>.json`

Rules and guidelines for runtime environment

Consider the following guidelines when you run tasks in different runtime environments:

- You cannot use a proxy server to connect to Informatica Intelligent Cloud Services when you use the Hosted Agent or the serverless runtime environment.
- You cannot connect to the REST API endpoints that require custom server certificate signed by CA and are not a part of Informatica cacerts truststore, when you use the Hosted Agent.
- You cannot configure JWT bearer token authentication when you use the Hosted Agent.
- Ensure that the swagger specification file URL is a public URL and returns the content of the file without prompting for further authentication and redirection when you use the Hosted Agent.

Rules and guidelines for a REST V2 connection

Consider the following rules and guidelines for a Rest V2 connection:

- When you test the connection, the Secure Agent validates the following parameters:
  - Path of the local Swagger file or the URL of the hosted Swagger file.
  - JSON format of the Swagger file.

  However, the Secure Agent does not validate endpoint credentials when you test the connection.
- You can configure proxy at the agent level or connection level. See the following table to understand the proxy settings that take precedence when you define the System proxy and proxy at the connection level:

System proxy	**REST V2 connection attribute**			**Result**
	No proxy	**Platform proxy**	**Custom proxy**	
No	Yes	No	No	Does not consider proxy.
No	No	Yes	No	Does not consider proxy.
No	No	No	Yes	Considers custom proxy.
Yes	Yes	No	No	Does not consider proxy.
Yes	No	Yes	No	Considers platform proxy.
Yes	No	No	Yes	Considers custom proxy.

CHAPTER 3

REST V2 operations

Create a mapping in the **Mapping Designer** to read or write data to the web service application.

When you create a mapping, you must select a REST V2 connection and an operation. The Secure Agent connects to the web service application to access, transform, or deliver data. If you want to write data to a relational target, the Secure Agent converts the hierarchical data fetched from the web service application to relational data. If you want to read data from a relational source and write data to a REST V2 target, the Secure Agent converts the relational data fetched from the source to hierarchical data.

REST V2 Connector can handle all HTTP 300 series status codes that indicate URL redirection in Source, Target, and Midstream transformations.

You can use REST V2 Connector to perform paging in Source and Midstream transformations.

REST V2 source operations

Create a Source transformation in the Mapping Designer to read data from the web service application.

When you select a REST V2 connection for a Source transformation in a mapping to read data from a web service application, specify Operations on the **Source** tab of the Source transformation. The Secure Agent displays operation IDs specified in the swagger specification.

Select an operation ID, configure the request message from the sample template provided in the request message editor, and configure the advanced properties for the operation. If you want to write to a relational target, define a relational structure for the source data by mapping the incoming fields that is in hierarchical format to the output fields in relational format. When you run the mapping, the Secure Agent retrieves data for the specified operation from the web service application.

Configuring a request using Request Message Editor

When you create a Source transformation, configure an XML request message for the operation that you want to perform in the web service application.

Use the **Request Message Editor** to create a request message. The request message is in XML format. You can use the sample request message for the operation and then customize the request message to specify the data that you want to enter into the data flow.

To customize your request, copy the request message from the sample template to the **Request Message Editor** pane where you can edit the XML message and add the attributes for the request. Remove unnecessary and empty tags from the request message to avoid operation failure.

If you have not defined the request body or parameters in the OpenAPI specification, the request message template displays a static field named Request_Port. Copy the request message from the sample template to the **Request Message Editor** pane where you can edit the XML message and add the attributes for the request.

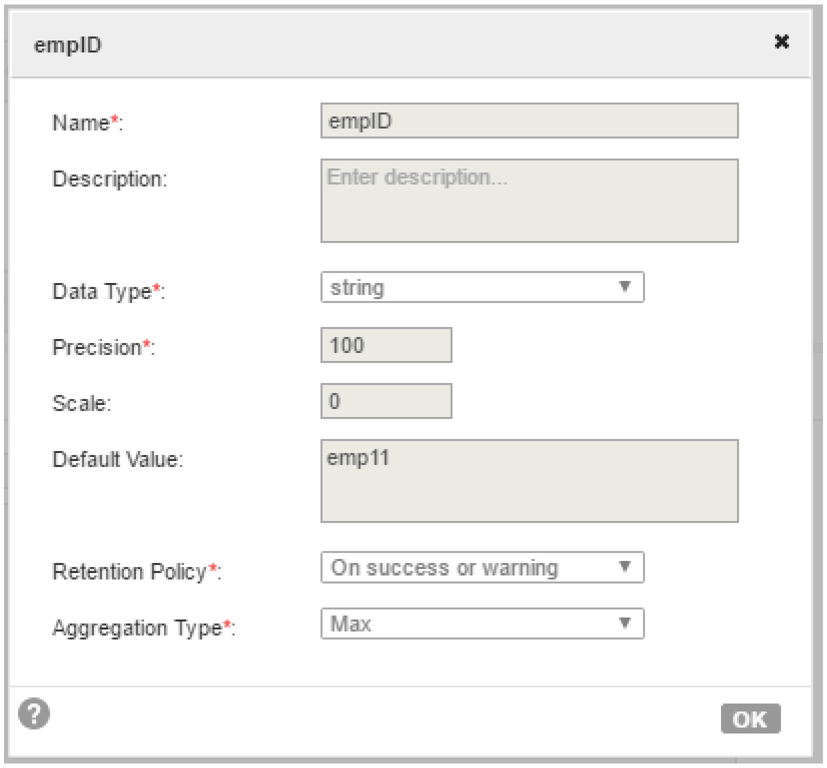
Header fields, path parameters, and query parameters appear as fields in source, target, midstream transformations.

Parameterizing the input values in a request message

You can use in-out parameters to represent input values in a request XML.

Configure the in-out parameters in the Mapping Designer. From the **Mapping Design** page, you open the parameters panel and configure an in-out parameter.

The following image shows a configured `empID` in-out parameter value:


empID

Name*:	empID
Description:	Enter description...
Data Type*:	string
Precision*:	100
Scale:	0
Default Value:	emp11
Retention Policy*:	On success or warning
Aggregation Type*:	Max

After you configure a parameter, use the parameter name in the following format, $$Name, in a request message. The XML uses the values of fields from the parameterized object.

For example, you want to use parameterized `empID` in an XML Request for a `CouchDB_INPUT` operation.

The following sample request shows the parameterized values that you can specify in an XML request:

```
<!--1 or more repetitions:-->
<proc:CouchDB_INPUT xmlns:proc="http://xml.schemas/infa/procedure/">
  <!--Optional:-->
  <CouchDB>
    <!--1 or more repetitions:-->
    <emp>
      $$empID
    </emp>
  </CouchDB>
</proc:CouchDB_INPUT>
```

Configuring values for in-out parameters in a mapping

You can use an in-out parameter that holds a variable value that can change each time a task runs to manage incremental data loading.

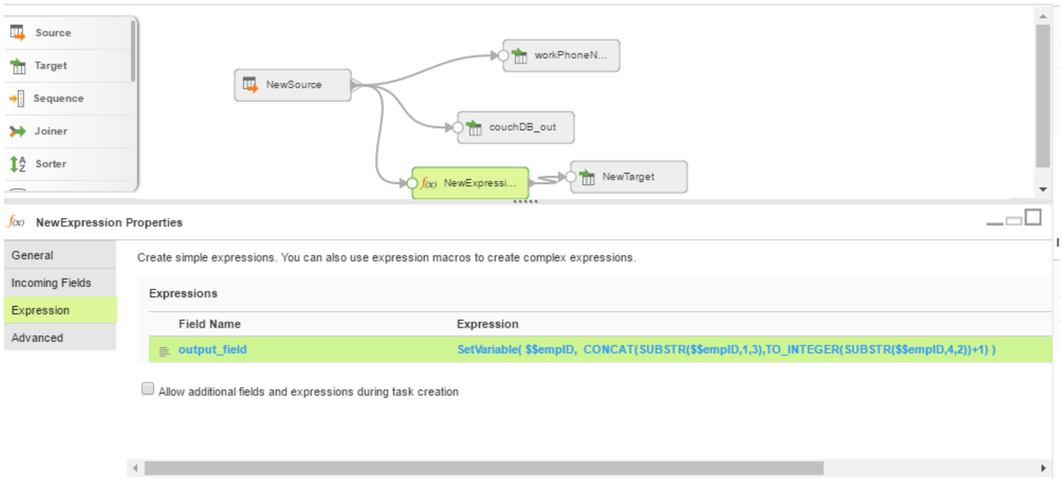
Add the in-out parameterized field in complex expressions in a mapping if you want to update the parameters when each task runs. The in-out parameter acts like a placeholder for a value that stores a task stage. Data Integration evaluates the parameter at runtime based on the specified configuration.

For example, you can use the following parameterized values in an Expression transformation:

```
SetVariable( $$empID, CONCAT(SUBSTR($$empID,1,3),TO_INTEGER(SUBSTR($$empID,4,2))+1) )
```

In the example, SetVariable function sets the parameter value each time the mapping task runs. You set a default value for the $$empID parameter. You want to update the value of $$empID by one every time the task runs.

The following image shows the logic used in expression transformation to assign value to in-out parameter in the Mapping Designer:



Create simple expressions. You can also use expression macros to create complex expressions.

Expressions

Field Name	Expression
output_field	SetVariable($$empID, CONCAT(SUBSTR($$empID,1,3),TO_INTEGER(SUBSTR($$empID,4,2))+1))

☐ Allow additional fields and expressions during task creation

When the task runs, the Secure Agent updates the in-out parameters based on the specified logic in the expression.

Field mapping in a source transformation

The response message format follows the service response definition in the Swagger or OpenAPI specification file. You can map response fields from a hierarchical to a relational structure of output groups and fields.

After you configure an operation for a source and specify the request message, create the relational format from the hierarchical data to include groups and fields that you want in the output.

To do this, select the elements in the response structure that you want to include as output fields. The Secure Agent converts the XML response in the hierarchical structure to relational groups at run time.

If you have not defined the response body in the swagger or OpenAPI specification, the field mapping displays a static field named Response_Port.

The field mapping displays static fields named OtherResponseheaders and OtherCookies that contain response headers and cookies.

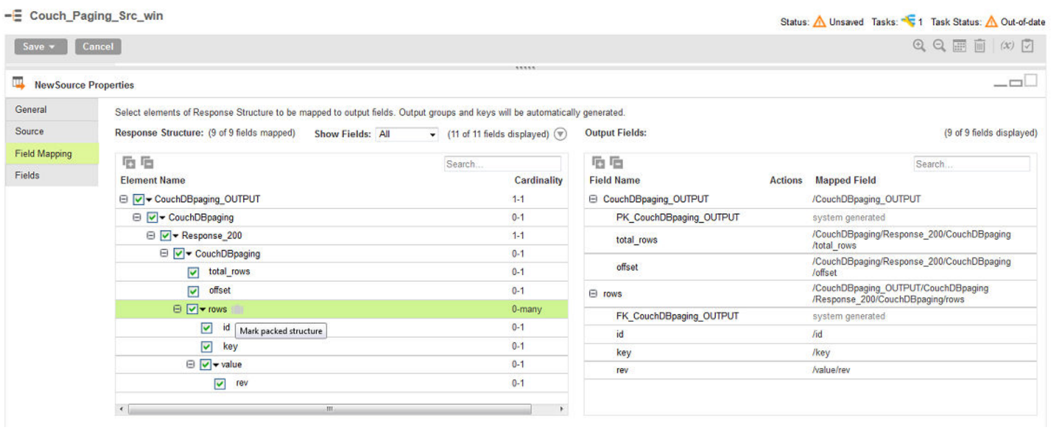
Creating packed fields

If you do not want to parse hierarchical elements, you can pack the elements into one field. You can also pack array elements when you write to the target.

A pack icon appears next to elements on the **Field Mapping** tab. When you select the pack icon that appears next to the element, the agent packs the element and its child elements into a single XML string during runtime. You cannot select fields that are child elements of packed fields in field mapping.

You can unpack fields by clicking on the pack icon.

To pack the fields, click on the pack icon that appears next to the rows element.



Select elements of Response Structure to be mapped to output fields. Output groups and keys will be automatically generated.

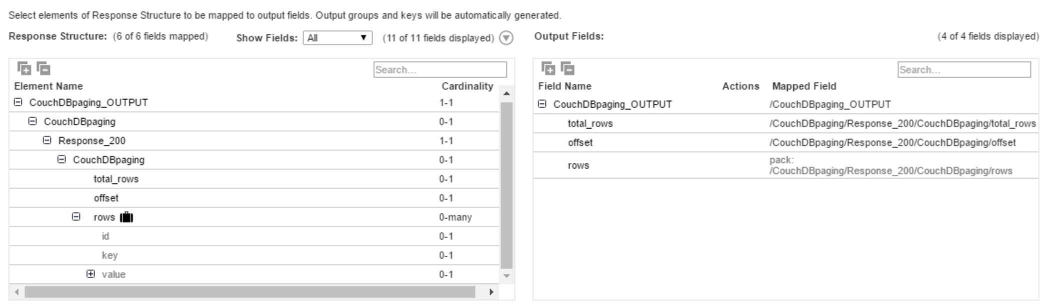
Response Structure: (9 of 9 fields mapped) Show Fields: All (11 of 11 fields displayed)

Element Name	Cardinality
CouchDBpaging_OUTPUT	1-1
CouchDBpaging	0-1
Response_200	1-1
CouchDBpaging	0-1
total_rows	0-1
offset	0-1
rows	0-many
id	0-1
key	0-1
value	0-1
rev	0-1

Output Fields: (9 of 9 fields displayed)

Field Name	Actions	Mapped Field
CouchDBpaging_OUTPUT		/CouchDBpaging_OUTPUT
PK_CouchDBpaging_OUTPUT		system generated
total_rows		/CouchDBpaging/Response_200/CouchDBpaging /total_rows
offset		/CouchDBpaging/Response_200/CouchDBpaging /offset
rows		/CouchDBpaging_OUTPUT/CouchDBpaging /Response_200/CouchDBpaging/rows
FK_CouchDBpaging_OUTPUT		system generated
id		/id
key		/key
rev		/value/rev

The following image shows the packed rows element:



Select elements of Response Structure to be mapped to output fields. Output groups and keys will be automatically generated.

Response Structure: (6 of 6 fields mapped) Show Fields: All (11 of 11 fields displayed)

Element Name	Cardinality
CouchDBpaging_OUTPUT	1-1
CouchDBpaging	0-1
Response_200	1-1
CouchDBpaging	0-1
total_rows	0-1
offset	0-1
rows	0-many
id	0-1
key	0-1
value	0-1

Output Fields: (4 of 4 fields displayed)

Field Name	Actions	Mapped Field
CouchDBpaging_OUTPUT		/CouchDBpaging_OUTPUT
total_rows		/CouchDBpaging/Response_200/CouchDBpaging/total_rows
offset		/CouchDBpaging/Response_200/CouchDBpaging/offset
rows		pack: /CouchDBpaging/Response_200/CouchDBpaging/rows

The following sample shows the output of the Rows field in the target file after you run the task:

```
<infa_packed>
<rows>
<id>52835a523b9b6816ec81e9585b09c987</id>
<key>52835a523b9b6816ec81e9585b09c987</key>
<value>
<rev>1-c7fe1b7b899a9ce50319a47d675af735</rev>
</value>
</rows>
</infa_packed>
```

REST V2 midstream operations

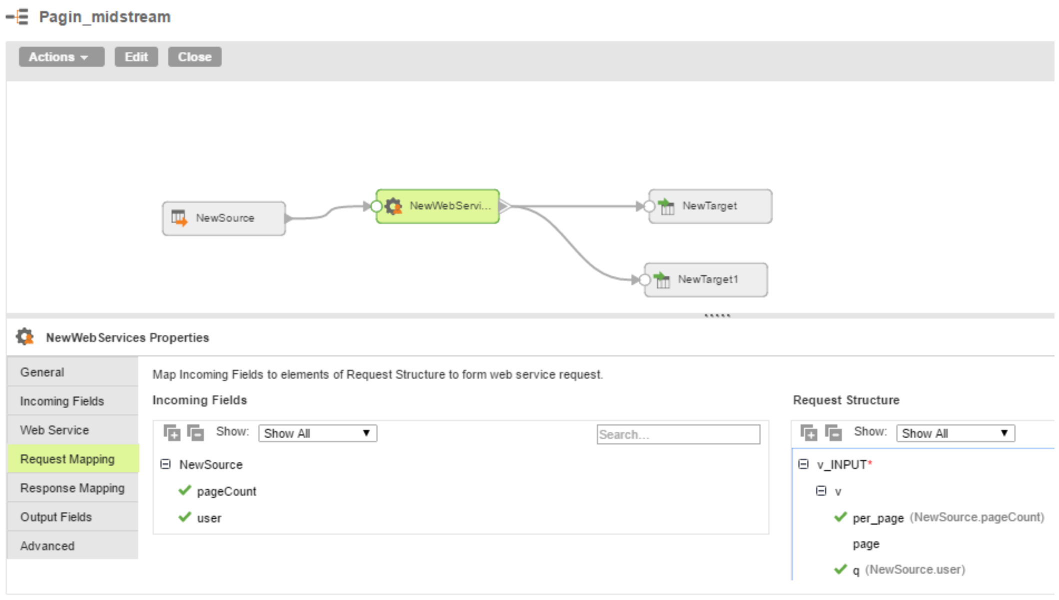
Create a REST V2 midstream transformation that interacts with the web service application to perform operations on hierarchical data.

When you use REST V2 Connector midstream, you create a business service, associate a REST V2 connection, and the required REST read or write operation for the business service. You can then use the business service to add operations to the Web Services transformation in the Mapping Designer.

Note: When you configure a business service, the values that appear in the **Operations** list are based on the operation IDs listed in the swagger specification file. If the operations IDs listed in the swagger specification file do not appear in the **Operations** list, contact Informatica Global Support.

A Web Services transformation connects to the web service application as a web service client to access, transform, or deliver data. Use the Web Services transformation in the Mapping Designer to construct a web service request and to parse the web service response.

The following image shows a REST V2 midstream transformation in a mapping:



Map Incoming Fields to elements of Request Structure to form web service request.

Incoming Fields — Show: Show All

- NewSource
  - ✓ pageCount
  - ✓ user

Request Structure — Show: Show All

- v_INPUT*
  - v
    - ✓ per_page (NewSource.pageCount)
    - page
    - ✓ q (NewSource.user)

On the Response Mapping tab of the Web Service transformation, the request generates a response structure for the specified REST operation from which you can select the elements that you require in the output. The output groups and keys generate automatically.

The response structure displays the fault output group by default. The fault group contains the request XML, error code, and error message. The fault group is displayed for the new business services. To map the fault group for the business services created in the previous version of connector, create new business services and import them again.

REST V2 target operations

Create a Target transformation in the Mapping Designer to write data to a web service application. When you select a REST V2 connection for a Target transformation in a mapping, the Secure Agent displays operation IDs specified in the swagger specification you configure in the connection. You must select an operation based on the object configured in the connection. When you want to write data that is in relational format to the web service application, the Secure Agent converts the relational data to a hierarchical format before writing to web service application.

When you configure a REST V2 target, you can select **Operations** on the **Target** tab to display the list of valid operations. The Secure Agent creates target fields based on the request message structure of the operation you select on the **Target** tab.

When you write data to a web service application, the Secure Agent includes all the mapped incoming fields from the source. You can add multiple input sources to write data to a web service application target object and define the primary and foreign key relationships between the input sources before you run the mapping. Each input source appears as separate group in the Field Mapping tab in the target transformation. You can define key relationships in the Field Mapping tab.

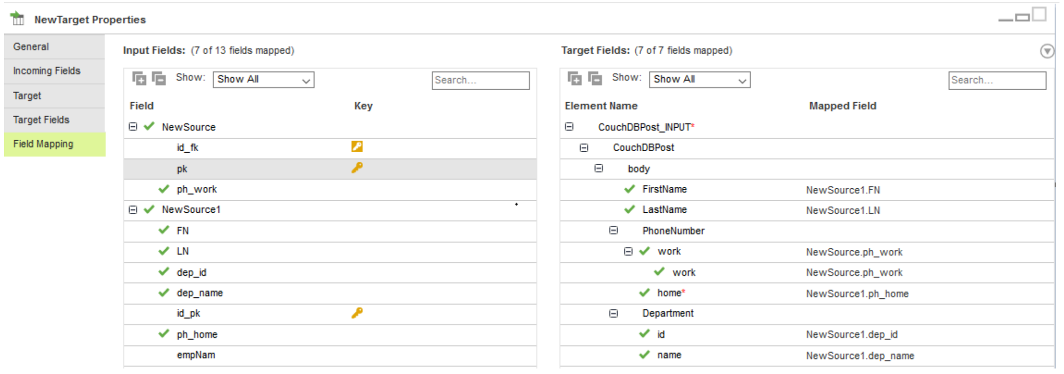
Field Mapping in a target transformation

You can map fields that are in a relational structure to the hierarchical structure used by the web service application.

On the **Field Mapping** tab for the Target transformation, the fields in the **Target Fields** appear in hierarchical format. The target fields are determined by the request message structure of the operation you select for the Target transformation.

Each source object displays as a group in the **Input Fields** area. You can select fields in the Input Fields area to map the fields to the request. If the input fields include multiple input groups, map the groups to the corresponding nodes in the request. Be sure to map all of the fields that you require in the request.

The following image shows an example of mapped input fields from the source file with the REST V2 target to update employee details:



Input Fields: (7 of 13 fields mapped)

Field	Key
⊟ ✓ NewSource	
id_fk	🔑
pk	🔑
✓ ph_work	
⊟ ✓ NewSource1	
✓ FN	
✓ LN	
✓ dep_id	
✓ dep_name	
id_pk	🔑
✓ ph_home	
empNam	

Target Fields: (7 of 7 fields mapped)

Element Name	Mapped Field
⊟ CouchDBPost_INPUT*	
⊟ CouchDBPost	
⊟ body	
✓ FirstName	NewSource1.FN
✓ LastName	NewSource1.LN
⊟ PhoneNumber	
⊟ ✓ work	NewSource.ph_work
✓ work	NewSource.ph_work
✓ home*	NewSource1.ph_home
⊟ Department	
✓ id	NewSource1.dep_id
✓ name	NewSource1.dep_name

Configuring a request that contains array elements

You can construct a request for a REST V2 target and REST V2 midstream transformation that contains array elements using one of the following methods:

- Create a request using multiple source groups
- Create a request using a single source group

Note: You cannot configure unnamed arrays in the request body.

Creating a request using multiple source groups

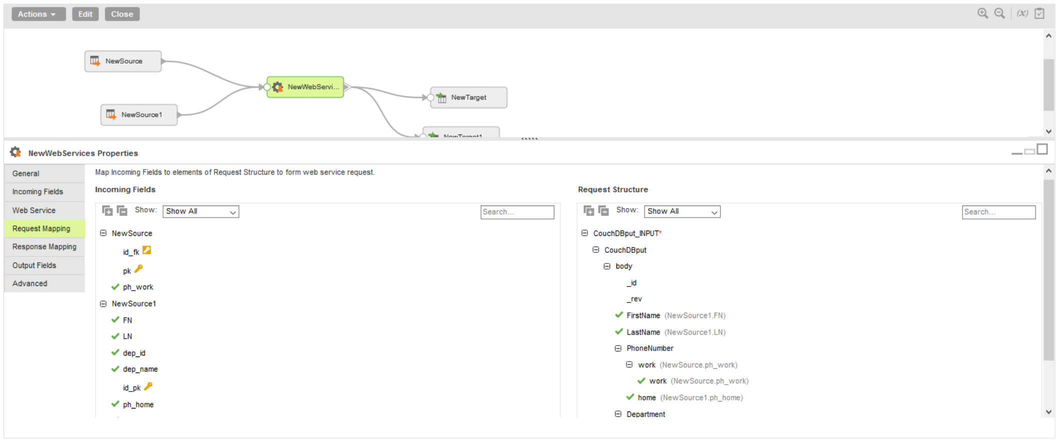
If the request message contains an element, for example, phone number of type work of an employee, that occurs multiple times, the input fields for this element must come from different source groups.

To create a request for relational sources, link the source group in the input fields to its parent source group by using the foreign key.

To decide the parent source group for multi-occurring elements, you must perform the following steps:

- The parent group must be the immediate multi-occurring parent element.
- In the absence of a multi-occurring parent element, map the multi-occurring element to the source group that is mapped to the root element.

For example, the following image shows the mapping of the multi-occurring element `work` from the relational data source to the REST V2 target:



Map Incoming Fields to elements of Request Structure to form web service request.

Incoming Fields

Show: Show All

```
⊟ NewSource
    id_fk
    pk
  ✓ ph_work
⊟ NewSource1
  ✓ FN
  ✓ LN
  ✓ dep_id
  ✓ dep_name
    id_pk
  ✓ ph_home
```

Request Structure

Show: Show All

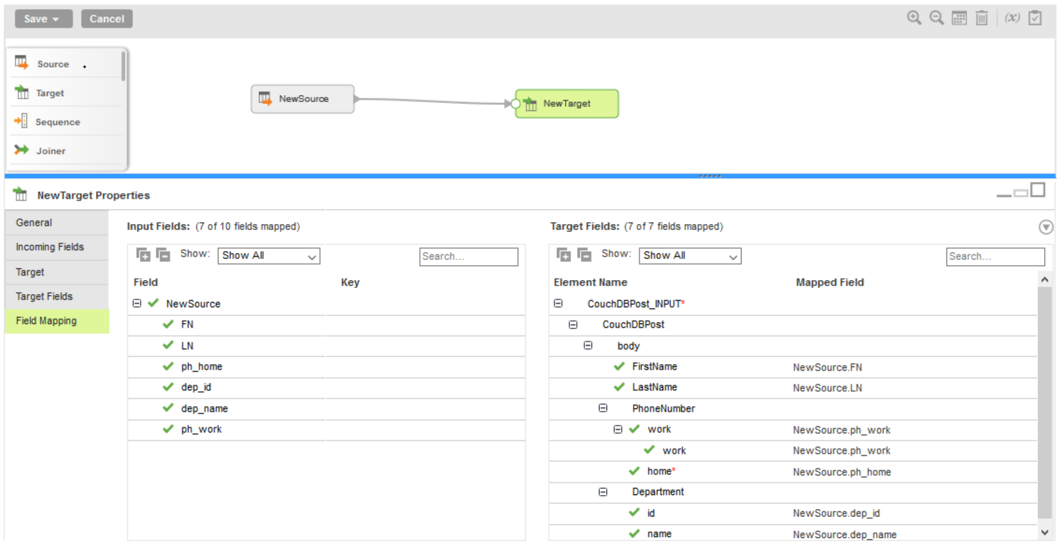
```
⊟ CouchDBput_INPUT*
  ⊟ CouchDBput
    ⊟ body
        _id
        _rev
      ✓ FirstName (NewSource1.FN)
      ✓ LastName (NewSource1.LN)
      ⊟ PhoneNumber
        ⊟ work (NewSource.ph_work)
          ✓ work (NewSource.ph_work)
        ✓ home (NewSource1.ph_home)
      ⊟ Department
```

Creating a request using a single source group

To create a request using a denormalized source for a midstream transformation or REST V2 target, perform the following steps:

1. Include all the fields in a single source file. Ensure that the multi-occuring elements are the last fields in the denormalized source file.
2. Click Administrator > Runtime Environments and select an agent.
3. Add a new property under **Custom Configuration Details**.
4. Select **Service** as Data Integration Server and **Type** as Tomcat.
5. Specify the name of the property as `ALLOW_DUPLICATE_VALUES`.
6. Set the value to `true`.
7. Click **Done** to save the changes.
8. Map the corresponding source to target fields.

For example, the following image shows the mapping of the field work to an array element of REST V2 target:



Input Fields: (7 of 10 fields mapped)

Field	Key
NewSource	
FN	
LN	
ph_home	
dep_id	
dep_name	
ph_work	

Target Fields: (7 of 7 fields mapped)

Element Name	Mapped Field
CouchDBPost_INPUT*	
CouchDBPost	
body	
FirstName	NewSource.FN
LastName	NewSource.LN
PhoneNumber	
work	NewSource.ph_work
work	NewSource.ph_work
home*	NewSource.ph_home
Department	
id	NewSource.dep_id
name	NewSource.dep_name

Uploading a file to a REST endpoint URL

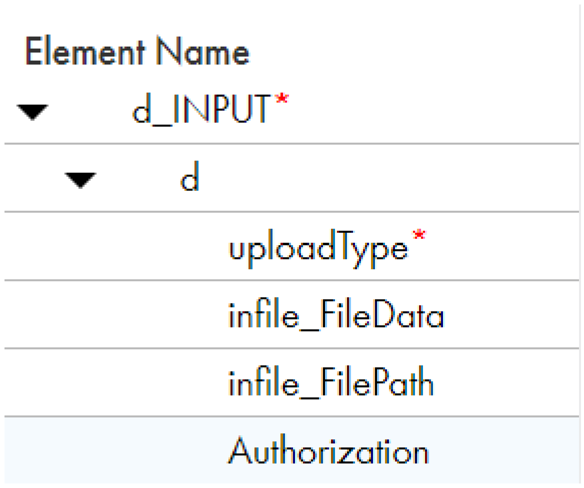
You can upload a file to a REST endpoint URL as a part of the REST API call.

To pass a file as an input to REST V2 connector, you must set the content-type to either `multipart/form-data` or `binary` in the request. When the swagger definition has Input parameter of type formData, the file boundary is added to the content of the uploaded file to indicate the start and end of the file. When the swagger definition has Input parameter of type `binary`, the content is generated without having the file boundary.

You must have the `formData` parameter of type `file` defined as one of input parameters in swagger. For example,

```
{ "name":"file",

"in":"formData",

"description":"file to upload",

"required":false,

"type":"file" }
```

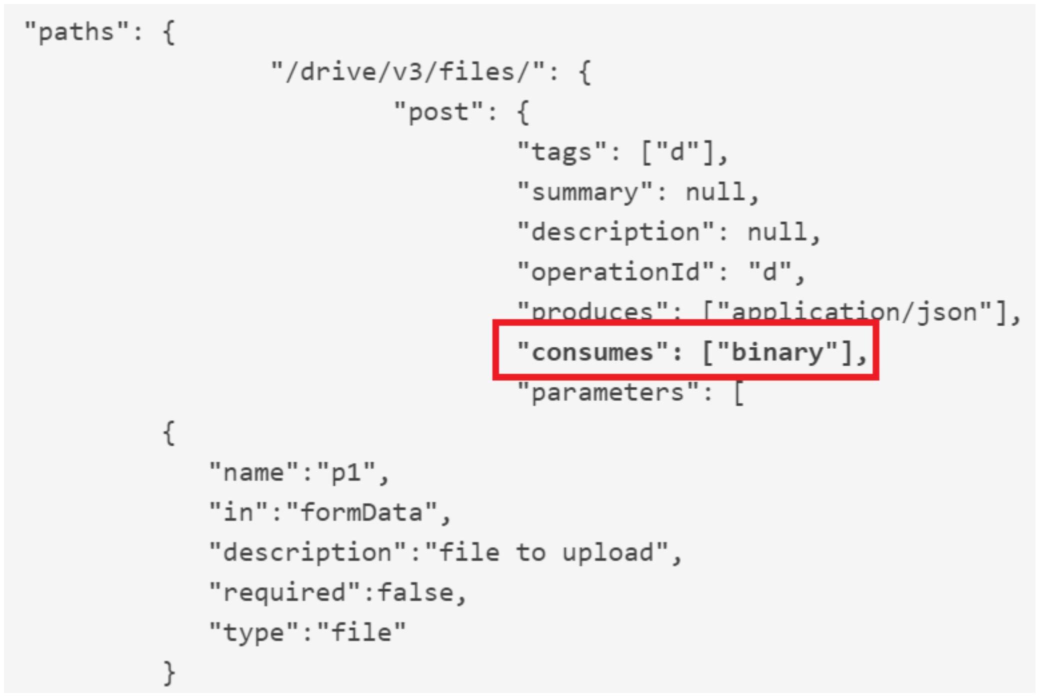
The following image shows the sample REST V2 object hierarchy:



Use one of the following methods to pass a file as an input:

- For the `xxx_FilePath` input field, specify the complete file path as value in the source. Used for any file formats.
- For the `xxx_FileData` input field, pass the Base64 encoded value of the file in the source. Used for file formats, such as `.pdf`, `.jpg`, `.xls`, and `.doc`. The length of the Base64 encoded value must not exceed 65535 characters.
- For the `xxx_FileData` input field, pass the file value as string in the source. Used for plain text file formats, such as `.txt`, `.JSON`, and `.xml`. The file size must not exceed 65535 characters.

The following image shows a sample swagger file to upload a binary file:



```
"paths": {
                "/drive/v3/files/": {
                        "post": {
                                "tags": ["d"],
                                "summary": null,
                                "description": null,
                                "operationId": "d",
                                "produces": ["application/json"],
                                "consumes": ["binary"],
                                "parameters": [
          {
             "name":"p1",
             "in":"formData",
             "description":"file to upload",
             "required":false,
             "type":"file"
          }
```

Parsing response headers

You can parse the header content in a response sent by a REST API endpoint.

Use the `headers` tag in the swagger file to parse the header content. By using the `Set-Cookie` tag, you can also parse the cookie content as part of the `header` tag. REST V2 Connector parses fields defined in the swagger file. Any extra fields coming from the web service for header or cookies appear under `OtherResponseHeaders` or `OtherCookies` fields respectively.

Defining Response Headers

```
"responses" : {

....

"headers": {

"places": {

"type": "string",

"description": "calls per hour allowed by the user"

},

"sessionId": {

"type": "integer",

"description": "date in UTC when token expires"

},

"Set-Cookie/activeSession": {

"type": "string",

"description": "cookies sample"

},

"Set-Cookie/PhoneId": {

"type": "integer",

"description": "cookies sample"

}

}
```

Note: The `Set-Cookie` tag is case-sensitive.

Parsing security definitions

You can define security definitions when you perform an operation on a REST API endpoint.

Define the `securityDefinitions` input parameter in the swagger file and call the security definitions in an operation. You can use only the basic and apiKey security definitions types with the `securityDefinitions` parameter.

Defining securityDefinitions

```
...
"security": [{"type": [],"id": [],"basic_key": []}]
}
}
},
"securityDefinitions": {
"type": {
"type": "apiKey",
"name": "type",
"in": "query",
"default":"json"
},
"id": {
"type": "apiKey",
"name": "id",
"in": "query"
},
"basic_key": {
"type": "basic",
"name": "basicAuth",
"in": "header"
}
}
```

If you want to connect to a web service that uses OAUTH version 2.0, provide the values of authentication details as custom header fields or query parameters in a request message.

Rules and guidelines for REST V2 operations

Consider the following rules and guidelines for REST V2 Operations:

- You cannot validate a request message.
- Elements related to header fields, path parameters, and query parameters appears as input fields in Source, Target, and Midstream transformations.
- You cannot configure a proxy server on which authentication is enabled.
- To parse xml elements in request and response structures, ensure that the element is enclosed within a parent element. For example:
  `<parentElement><childElement>xyz</childElement></parentElement>`

- You can only use Integer, Decimal, and Long data types. Set type as normal and format as int64 for fields that contain values of Long data type in the swagger specification file.
- You can use packed field only in a Source transformation.
- You cannot use extended ASCII characters and I18N characters in a field name for JSON payload.
- You cannot use special characters in the field name for query parameters and operation names.
- You must specify operationId for all operations in the OpenAPI specification file.
- You cannot use special characters in the array type fields.
- When you use application/xml or application/x-www-form-urlencoded media type to process data, do not use special characters, such as &, >, and < in data for headers, body, query parameters, and input fields. These characters are transformed in the target API to an encoded format. For example, & is encoded as &amp.
- If you use invalid XML tags or special characters, such as &, >, and < in body and input fields in the application/xml media type, the task throws an exception during XML validation. For example, if input data is `<BODID>infor-nid:infor:15101113565:?Employee&verb=Sync</BODID>`, the following error message appears based on the input data:
  ```
  [ERROR][Fatal Error] :1:516: The reference to entity "verb" must end with the ';'
  delimiter
  ```
  You can ignore the error message.
- You must not create in-out parameters for advanced properties, such as Page Parameter, Start Page, End Page, and Override URL.
- REST V2 Connector overrides the Content-Type and Accept headers.
- If an endpoint API path contains spaces, you must define the path as path parameter.
- If the swagger URL you specify is not accessible or the proxy server is not accessible, mappings fail and one of the following errors is logged in the session log. The behavior is same for the source, midstream, and target mappings.
  - `[ERROR] com.informatica.sdk.helper.common.ApiException: java.net.UnknownHostException: inv28pers108Communication exception, Proxy settings might be incorrect.`
  - `[ERROR] com.informatica.sdk.helper.common.ApiException: java.net.ConnectException: Connection refused: connectCommunication exception, Proxy settings might be incorrect.`
- You can define a page parameter on the request input field if the payload is of type JSON. The page parameter gets resolved only when the JSON request payload is an unnamed object. For example:
  ```
  "body" : {
  "a" : "ty",
  "b" : 0
  }
  ```
  The page parameter does not get resolved in the following scenarios:
  - When the JSON request payload is a named object.
  - The parameter defined is a nested input element.
  - The parameter defined is an array element.
  - When the payload is of type XML.
- You can define default values for request input parameters except for the `body` and `securityDefinitions` parameters. For example:
  ```
  "parameters": {
  "keyReference": {
  ```

```
"name": "keyReference",

"in": "header",

"required": false,

"type": "integer",

"default": 123

}

}
```

Default values are applicable to integer, string, boolean, and array of primitive types fields.

- When you use the REST V2 Connector as source, the default values defined in the swagger file are not honored if you pass empty values for input parameters in the request message editor. If you do not provide values in request message fields and want the fields to pick up default values defined in the swagger definition, you must remove the fields from the request message editor.
- If the default value for an input parameter is defined as blank, the blank value is treated as NULL unless you set the value explicitly in a mapping.
- You cannot define XML attributes using inline elements in the swagger file. Define XML attributes as separate elements in the swagger file and refer the XML attributes to extract data.
- When you use a REST V2 connection to configure a source or target operation and try to modify the object by manually typing a valid name in the **Operation** field, the task fails with the following validation error: `Error occurred validating object: Selected connection does not support this operation.`

  Workaround: Select the same operation from the **Select...** dialog box and run the task again.
- When you read JSON data with an array of response, the session log contains only the first instance of the array.
- When you define parameters in the request body, ensure that the name under parameters and definitions is the same. In the following example, if the parameter name is **getAccounts**, the definition name in $ref must also be **getAccounts**:

  ```
  "parameters": {
  "name": "getAccounts",
  "in": "body",
  "required": true,
  "schema":{
  "$ref":"#/definitions/getAccounts"
   }
  ```
- You can either define an attribute or a value in a single XML element. The following example where a single XML element contains both an attribute and a value is not applicable:

  ```
  <wd:ID wd:type="Organization_Role">Manager</wd:ID>
  ```

CHAPTER 4

Mappings and mapping tasks with REST V2 Connector

Use the Data Integration Mapping Designer to create a mapping. When you create a mapping, you configure a source or target to represent a Rest V2 object. In advanced mode, the Mapping Designer updates the mapping canvas to include transformations and functions that enable advanced functionality.

REST source transformation in mappings

When you configure a Source transformation, select the REST V2 connection and choose an operation to represent a web service source.

You can select an operation for the source through the connection. Configure the request message using the request message template. You can parameterize the input values in the request XML. Configure the paging attributes.

You can view the response structure in the field mapping. When you map the elements from the response structure to the output fields, the Secure Agent creates the output groups, along with the primary and foreign keys for the field names. When you deploy the mapping in a mapping task and run the task, the Secure Agent reads the data from web service.

Advanced source properties

In a mapping, you can configure a source to represent a web service application source. For the REST source connections used in a mapping, you can configure advanced properties in the **Source** tab in the Mapping Designer.

The following table describes the advanced properties that you can configure in a source:

Property	Description
Paging Type	Specify one of the following values: **Page**. Enables paging and considers the values of Page Parameter, Start Page, End Page, and End of Response Expression properties. **None**. Ignores the values of Page Parameter, Start Page, End Page, and End of Response Expression properties.
Page Parameter	The name of the parameter that you want to use for the paging operation. You can use a query parameter or a path parameter. The parameter must be of the integer type and from the request message.
Start Page	The page number that indicates the first page in the range, on which you want to perform the paging operation.
End Page	The page number that indicates the last page in the range, on which you want to perform the paging operation. The default is 10000. Paging stops when the End Page is reached or the End of Response Expression is met.
Page Increment Factor	An integer to increment the Page Parameter. Page Increment Factor must be same as the number of records being fetched per request. If the Page Increment Parameter option is not same as the number of records being fetched per request, you might have missing or duplicate records between two calls.
End of Response Expression	Specify an expression or a string to control paging. You can observe one of the following behaviors: - If you specify a string or an expression, the paging stops when the value matches with the page response. It does not parse the page that has matching end of response. - If you do not specify a string or an expression, the paging stops on reaching a page that has an empty or a Null response. If an empty or a Null response is never reached, the paging stops at the default end page. - If you specify both, End Page and End of Response Expression, the paging stops on whichever condition is met first. - If you do not specify End Page and End of Response Expression, the paging stops on reaching an empty or a Null response. If an empty or a Null response is never reached, the paging stops at the default end page.
Override URL	Overrides the URL specified in the swagger specification. The override URL cannot have query parameters. When a path parameter is included in the Override URL, enclose the path parameter with curly brackets {}. For example: URL specified in the swagger specification: `http://invr28pers102:13080/sample/day/20170505?a:b` If you define 20170505 as a path variable with the path variable name as `path1`, the Override URL will be as follows: `http://invr28pers102:13080/sample/day/{path1}`

Property	Description
Tracing Level	Amount of detail that appears in the log for the source. Use the following tracing levels: - Terse - Normal - Verbose Initialization - Verbose Default is normal.
Cache Size for Web Service Response (KB)	Memory available for the web service response. If the web service response contains many rows or columns, you might want to increase the cache size. Default is 1024 KB. Maximum is 99999 KB.

Sample End of Response Expression

Use End of Response Expression for paging. The following snippet shows a sample response for a page :

```
{

"code": "SUCCESS",

"validationResult": [],

"systemErrors": [],

"patientResponseData": [],

"count": 0,

"message": "Unable to retrieve the implant details"

}
```

For End of Response Expression, you can use the string, Unable to retrieve the implant details. If you want to match multiple conditions in the response page, you can use the following expression:

```
(.*)"patientResponseData": [](.*)Unable to retrieve the implant details
```

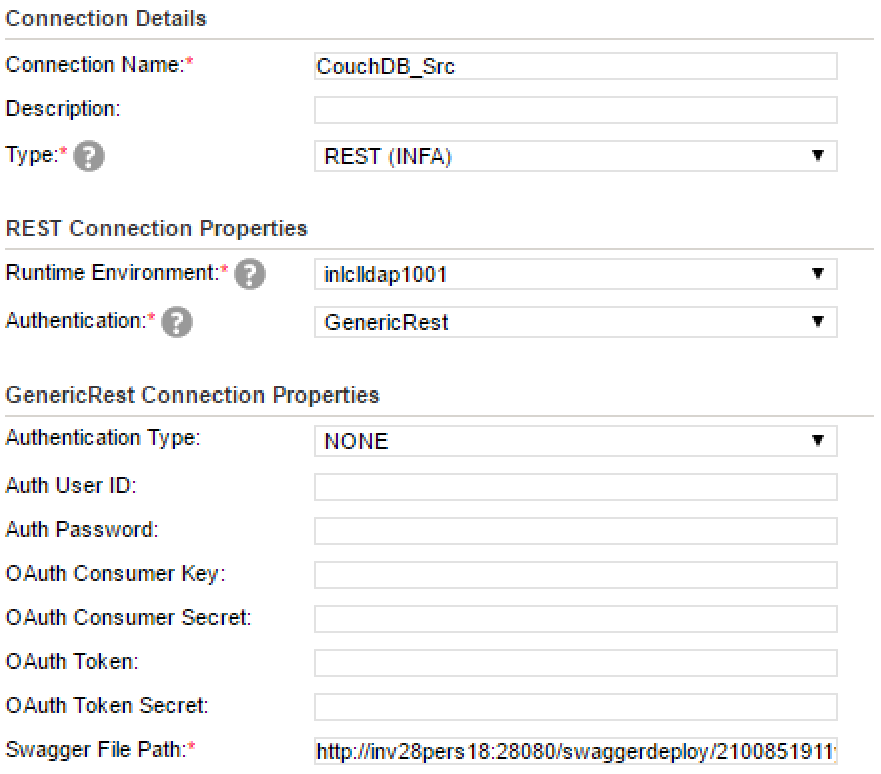
The above expression ensures that the paging will stop when both "patientResponseData": [] and Unable to retrieve the implant details are matched in the page response.

REST source mapping example

You are a human resources administrator and you want to extract contact information, such as first name, last name, email, and phone number of employees from Apache CouchDB to a flat file.

1. Create a REST connection. Verify that you specify the absolute path or the hosted URL of the swagger specification file, and the authentication method in the connection properties.

   The following image shows the configured REST connection:



Connection Details

Connection Name:*	CouchDB_Src
Description:	
Type:*	REST (INFA)

REST Connection Properties

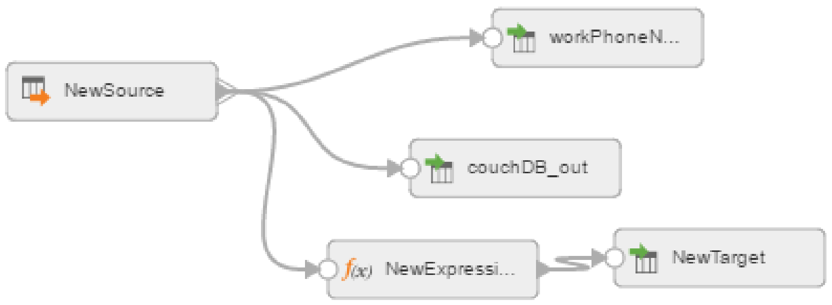
Runtime Environment:*	inlclldap1001
Authentication:*	GenericRest

GenericRest Connection Properties

Authentication Type:	NONE
Auth User ID:	
Auth Password:	
OAuth Consumer Key:	
OAuth Consumer Secret:	
OAuth Token:	
OAuth Token Secret:	
Swagger File Path:*	http://inv28pers18:28080/swaggerdeploy/2100851911

2. Create a flat file connection to write data to the flat file.
3. Create a REST mapping.

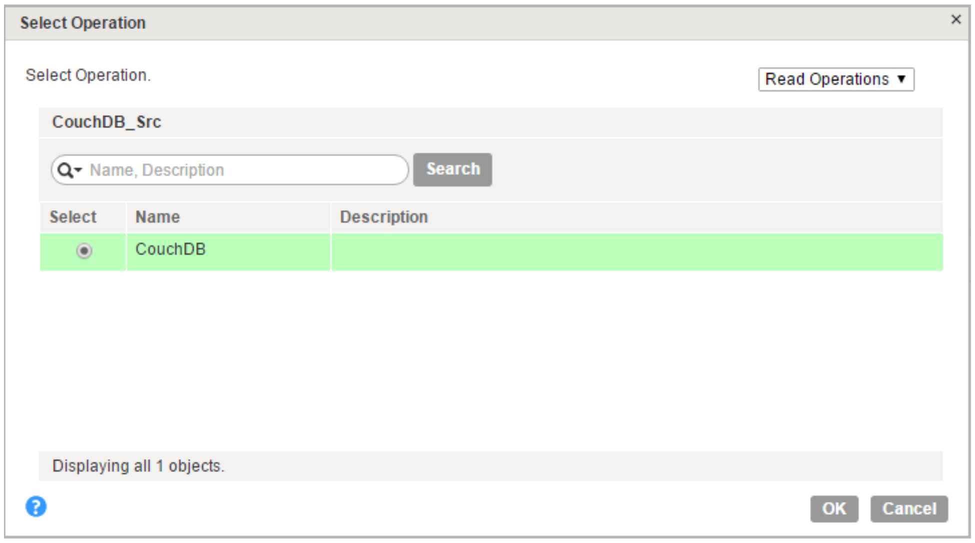
   The following image shows the REST mapping:



4. Add a Source transformation. Specify a name and description in the general properties.
5. On the **Source** tab, perform the following steps:
   a. In the **Connection** field, select the configured REST connection to connect to the Couch database.

b. In the Operation field, select CouchDB as the operation.

   The following image shows the CouchDB operations:



Select Operation

Select Operation. Read Operations

CouchDB_Src

Select	Name	Description
●	CouchDB	

Displaying all 1 objects.

c. In the **Request Options** section, configure the request message in the following XML format, specify the attributes in the message, and validate the message:

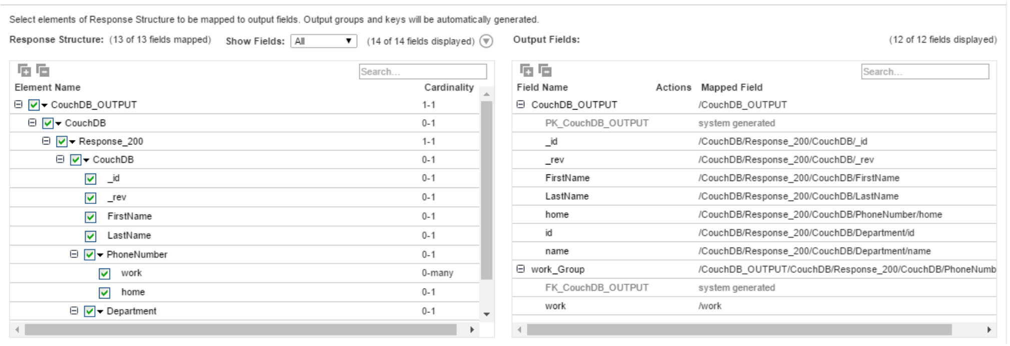
```
<!--1 or more repetitions:-->
<proc:CouchDB_INPUT xmlns:proc="http://xml.schemas/infa/procedure/">
  <!--Optional:-->
  <CouchDB>
    <!--1 or more repetitions:-->
    <emp>
      $$empID
    </emp>
  </CouchDB>
</proc:CouchDB_INPUT>
```

   The request message fetches the details of an employee based on the specified employee ID.

d. In the **Advanced Properties** section, set the tracing level to Normal, and use the default cache size of 1024 KB.

6. On the Field Mapping tab, select the elements, such as `_id`, `FirstName`, and `LastName` in the response structure that you want to map to the output fields.

   The following image shows the response structure on the left pane in a hierarchical format and the output groups on the right pane in a relational format:



Select elements of Response Structure to be mapped to output fields. Output groups and keys will be automatically generated.

Response Structure: (13 of 13 fields mapped) Show Fields: All (14 of 14 fields displayed)

Element Name	Cardinality
CouchDB_OUTPUT	1-1
CouchDB	0-1
Response_200	1-1
CouchDB	0-1
_id	0-1
_rev	0-1
FirstName	0-1
LastName	0-1
PhoneNumber	0-1
work	0-many
home	0-1
Department	0-1

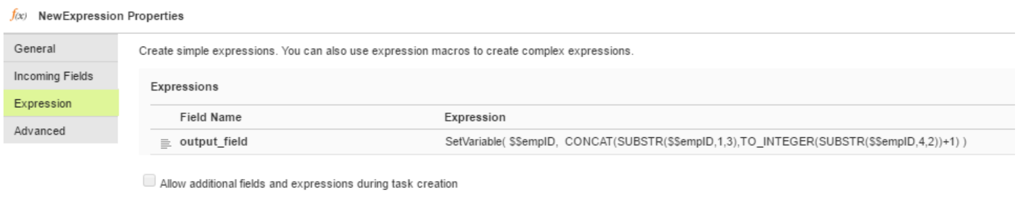
Output Fields: (12 of 12 fields displayed)

Field Name	Actions	Mapped Field
CouchDB_OUTPUT		/CouchDB_OUTPUT
PK_CouchDB_OUTPUT		system generated
_id		/CouchDB/Response_200/CouchDB/_id
_rev		/CouchDB/Response_200/CouchDB/_rev
FirstName		/CouchDB/Response_200/CouchDB/FirstName
LastName		/CouchDB/Response_200/CouchDB/LastName
home		/CouchDB/Response_200/CouchDB/PhoneNumber/home
id		/CouchDB/Response_200/CouchDB/Department/id
name		/CouchDB/Response_200/CouchDB/Department/name
work_Group		/CouchDB_OUTPUT/CouchDB/Response_200/CouchDB/PhoneNumb
FK_CouchDB_OUTPUT		system generated
work		/work

   The Secure Agent creates two output groups, workPhoneNumber and couchDB_out, which results in two relational output files. Primary and foreign keys are auto-generated.

7. Add three Target transformations and specify the target objects for each of the transformations. Perform the following steps:
    a. Select a flat file connection for each of the target transformations to write data to the flat files.
    b. Create target objects workPhoneNumber.csv, couchDB_out.csv, and NewTarget.csv files to write data to the target.
    c. Select the output group from CouchDB that you want to link to the target objects.
8. Add an Expression transformation, and include a `SetVariable` function and the parameterized value `$$empID` in the expression so that the task updates the value of `empID` by one at the end of each run.

    The following image shows the configured expression that utilizes the in-out parameters:

    NewExpression Properties

    General | Incoming Fields | Expression | Advanced

    Create simple expressions. You can also use expression macros to create complex expressions.

    Expressions

Field Name	Expression
output_field	SetVariable($$empID, CONCAT(SUBSTR($$empID,1,3),TO_INTEGER(SUBSTR($$empID,4,2))+1))

    ☐ Allow additional fields and expressions during task creation

9. Click **New** > **Tasks** > **Mapping Task**, and select the mapping for the task.
10. When you save and run the mapping task, the Secure Agent retrieves employee records from CouchDB and writes the data to the corresponding flat files.

REST midstream transformation in mappings

Before you configure a midstream transformation, you must create a business service from the **New** > **Components** tab. You must select a REST V2 connection and an operation when you create a business service.

When you configure the midstream transformation, select the business service and the operation on the **Web Service** tab.

You can map the input fields from source file to elements in request structure on the **Request Mapping** tab.

You can view the response structure in the field mapping. When you map the elements from the response structure to the output fields, the Secure Agent creates the output groups, along with the primary and foreign keys for the field names.

Advanced midstream properties

The following table describes the advanced properties that you can configure for a midstream transformation:

Property	Description
Paging Type	Specify one of the following values: **Page**. Enables paging and considers the values of Page Parameter, Start Page, End Page, and End of Response Expression properties. **None**. Ignores the values of Page Parameter, Start Page, End Page, and End of Response Expression properties.
Page Parameter	The name of the parameter that you want to use for the paging operation. You can use a query parameter or a path parameter. The parameter must be of the integer type and from the request message.
Start Page	The page number that indicates the first page in the range, on which you want to perform the paging operation.
End Page	The page number that indicates the last page in the range, on which you want to perform the paging operation. The default is 10000. Paging stops when the End Page is reached or the End of Response Expression is met.
Page Increment Factor	An integer to increment the Page Parameter. Page Increment Factor must be same as the number of records being fetched per request. If the Page Increment Parameter option is not same as the number of records being fetched per request, you might have missing or duplicate records between two calls.
End of Response Expression	Specify an expression or a string to control paging. You can observe one of the following behaviors: - If you specify a string or an expression, the paging stops when the value matches with the page response. It does not parse the page that has matching end of response. - If you do not specify a string or an expression, the paging stops on reaching a page that has an empty or a Null response. If an empty or a Null response is never reached, the paging stops at the default end page. - If you specify both, End Page and End of Response Expression, the paging stops on whichever condition is met first. - If you do not specify End Page and End of Response Expression, the paging stops on reaching an empty or a Null response. If an empty or a Null response is never reached, the paging stops at the default end page.
Override URL	Overrides the URL specified in the swagger specification. The override URL cannot have query parameters. When a path parameter is included in the Override URL, enclose the path parameter with curly brackets {}. For example: URL specified in the swagger specification: `http://invr28pers102:13080/sample/day/20170505?a:b` If you define 20170505 as a path variable with the path variable name as `path1`, the Override URL will be as follows: `http://invr28pers102:13080/sample/day/{path1}`
Cache Size for Web Service Request (KB)	Default is 100 KB. If the request is more than 100 KB, you can increase the cache size.
Cache Size for Web Service Response (KB)	Memory available for the web service response. If the web service response contains many rows or columns, you might want to increase the cache size. Default is 100 KB.

Property	Description
Allow Input Flush	Not Applicable.
Transaction Commit Control	Not Applicable.

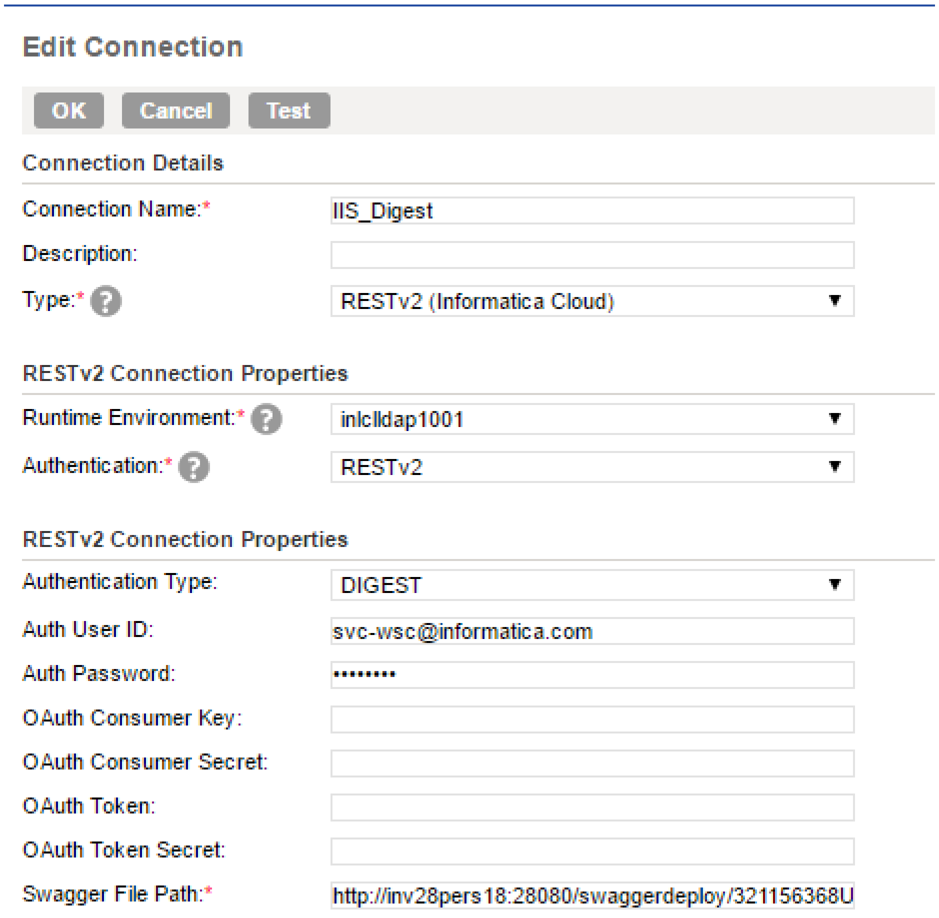
Midstream transformation mapping example

You are a human resources administrator and you want to retrieve details of an employee from Apache CouchDB.

To retrieve employee details from CouchDB and to write the employee details to a flat file, perform the following tasks:

1. Create a REST V2 connection to read data from CouchDB.

   The following image shows the configured CouchDB connection:



Connection Details

Connection Name:*	IIS_Digest
Description:	
Type:*	RESTv2 (Informatica Cloud)

RESTv2 Connection Properties

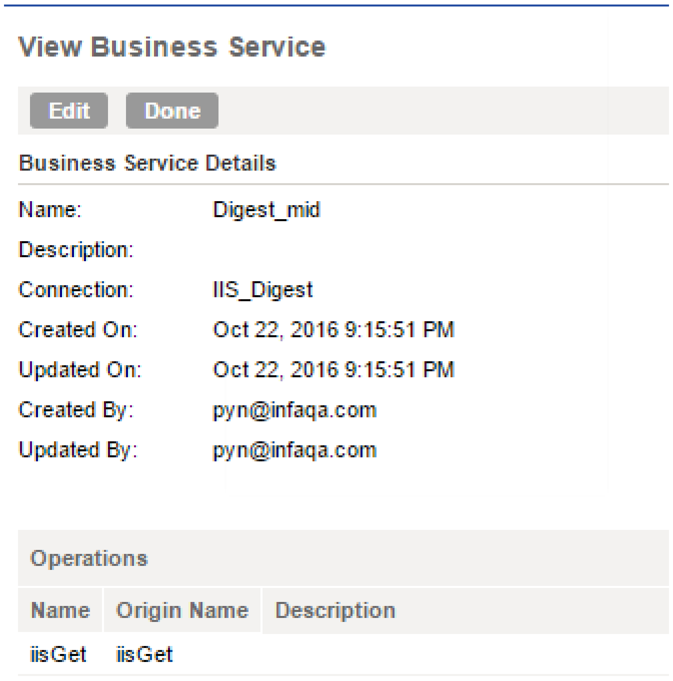
Runtime Environment:*	inlclldap1001
Authentication:*	RESTv2

RESTv2 Connection Properties

Authentication Type:	DIGEST
Auth User ID:	svc-wsc@informatica.com
Auth Password:	••••••••
OAuth Consumer Key:	
OAuth Consumer Secret:	
OAuth Token:	
OAuth Token Secret:	
Swagger File Path:*	http://inv28pers18:28080/swaggerdeploy/321156368U

2. Create a business service to associate the REST V2 connection and an operation.

The following image shows the configured business service:



View Business Service

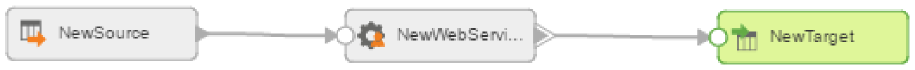
Business Service Details

Name:	Digest_mid
Description:	
Connection:	IIS_Digest
Created On:	Oct 22, 2016 9:15:51 PM
Updated On:	Oct 22, 2016 9:15:51 PM
Created By:	pyn@infaqa.com
Updated By:	pyn@infaqa.com

Operations

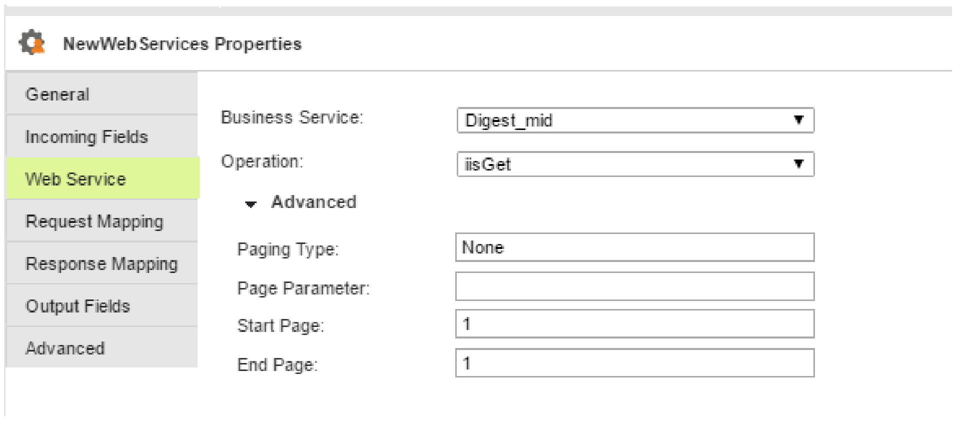
Name	Origin Name	Description
iisGet	iisGet	

3. Create a REST V2 mapping.



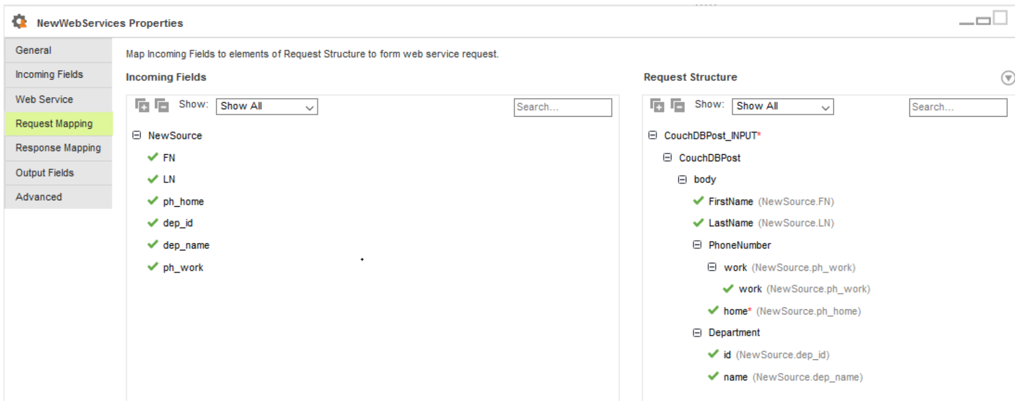
4. Add a Web Services transformation. Specify a name and description in the general properties.
5. On the **Web Service** tab, select the business service and the operation that you configured. Also, specify paging parameters.

   The following image shows the configured web service:



Business Service:	Digest_mid
Operation:	iisGet
Advanced	
Paging Type:	None
Page Parameter:	
Start Page:	1
End Page:	1

6. On the **Request Mapping** tab, map the incoming fields from the source to the respective fields in CouchDB.



NewWebServices Properties

General | Incoming Fields | Web Service | Request Mapping | Response Mapping | Output Fields | Advanced

Map Incoming Fields to elements of Request Structure to form web service request.

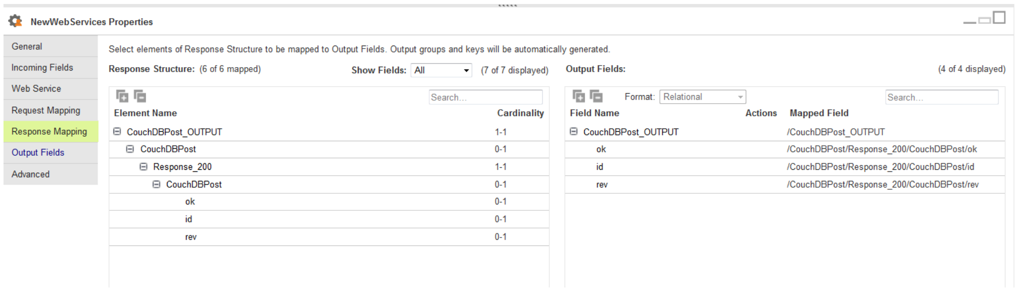
Incoming Fields — Show: Show All

- NewSource
  - ✓ FN
  - ✓ LN
  - ✓ ph_home
  - ✓ dep_id
  - ✓ dep_name
  - ✓ ph_work

Request Structure — Show: Show All

- CouchDBPost_INPUT*
  - CouchDBPost
    - body
      - ✓ FirstName (NewSource.FN)
      - ✓ LastName (NewSource.LN)
      - PhoneNumber
        - work (NewSource.ph_work)
          - ✓ work (NewSource.ph_work)
        - ✓ home* (NewSource.ph_home)
      - Department
        - ✓ id (NewSource.dep_id)
        - ✓ name (NewSource.dep_name)

7. On the **Response Mapping** tab, select the employee details fields that you want to write to the target file on the **Response Structure**.



NewWebServices Properties

General | Incoming Fields | Web Service | Request Mapping | Response Mapping | Output Fields | Advanced

Select elements of Response Structure to be mapped to Output Fields. Output groups and keys will be automatically generated.

Response Structure: (6 of 6 mapped) Show Fields: All (7 of 7 displayed)

Element Name	Cardinality
CouchDBPost_OUTPUT	1-1
CouchDBPost	0-1
Response_200	1-1
CouchDBPost	0-1
ok	0-1
id	0-1
rev	0-1

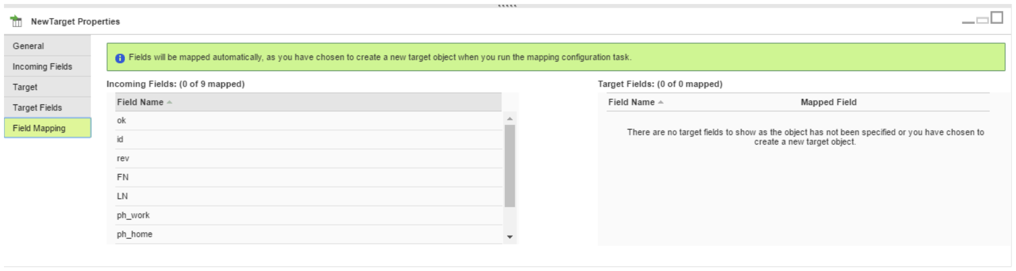
Output Fields: (4 of 4 displayed) Format: Relational

Field Name	Actions	Mapped Field
CouchDBPost_OUTPUT		/CouchDBPost_OUTPUT
ok		/CouchDBPost/Response_200/CouchDBPost/ok
id		/CouchDBPost/Response_200/CouchDBPost/id
rev		/CouchDBPost/Response_200/CouchDBPost/rev

8. On the **Advanced** tab, specify the cache size details.

   The fields that you selected from the **Response Structure** of the Web Service transformation appear as incoming fields for the target object.

9. If required, map the incoming fields to the flat file fields.



NewTarget Properties

General | Incoming Fields | Target | Target Fields | Field Mapping

Fields will be mapped automatically, as you have chosen to create a new target object when you run the mapping configuration task.

Incoming Fields: (0 of 9 mapped)

Field Name
ok
id
rev
FN
LN
ph_work
ph_home

Target Fields: (0 of 0 mapped)

Field Name	Mapped Field

There are no target fields to show as the object has not been specified or you have chosen to create a new target object.

10. Click **New** > **Tasks** > **Mapping Task**, and select the mapping for the task.

11. When you save and run the mapping task, the Secure Agent retrieves the employee record for the employee that you map on the Request Mapping tab from CouchDB and writes the data to the corresponding flat files.

REST V2 targets in mappings

When you select a REST V2 connection for a Target transformation, you can select an operation. The operation is based on the swagger specification file that you specify during the REST V2 connection configuration.

You can add multiple input groups into the REST target and define the primary and foreign key relationships between the multiple input groups before the mapping.

Advanced target properties

In a mapping, you can configure a target to represent a REST target. For REST target connections used in a mapping, you can configure advanced properties in the **Targets** page of the Mapping Task wizard.

The following table describes the advanced properties that you can configure in a Target transformation:

Property	Description
Paging Type	Specify one of the following values: **Page**. Enables paging and considers the values of Page Parameter, Start Page, End Page, and End of Response Expression properties. **None**. Ignores the values of Page Parameter, Start Page, End Page, and End of Response Expression properties.
Page Parameter	The name of the parameter that you want to use for the paging operation. You can use a query parameter or a path parameter. The parameter must be of the integer type and from the request message.
Start Page	The page number that indicates the first page in the range, on which you want to perform the paging operation.
End Page	The page number that indicates the last page in the range, on which you want to perform the paging operation. The default is 10000. Paging stops when the End Page is reached or the End of Response Expression is met.
Page Increment Factor	An integer to increment the Page Parameter. Page Increment Factor must be same as the number of records being fetched per request. If the Page Increment Parameter option is not same as the number of records being fetched per request, you might have missing or duplicate records between two calls.
End of Response Expression	Specify an expression or a string to control paging. You can observe one of the following behaviors: - If you specify a string or an expression, the paging stops when the value matches with the page response. It does not parse the page that has matching end of response. - If you do not specify a string or an expression, the paging stops on reaching a page that has an empty or a Null response. If an empty or a Null response is never reached, the paging stops at the default end page. - If you specify both, End Page and End of Response Expression, the paging stops on whichever condition is met first. - If you do not specify End Page and End of Response Expression, the paging stops on reaching an empty or a Null response. If an empty or a Null response is never reached, the paging stops at the default end page.

Property	Description
Override URL	Overrides the URL specified in the swagger specification. The override URL cannot have query parameters. When a path parameter is included in the Override URL, enclose the path parameter with curly brackets {}. For example: URL specified in the swagger specification: `http://invr28pers102:13080/sample/day/20170505?a:b` If you define 20170505 as a path variable with the path variable name as `path1`, the Override URL will be as follows: `http://invr28pers102:13080/sample/day/{path1}`
Cache Size for Web Service Request (KB)	Memory available for the web service request. If the web service request contains many rows or columns, you might want to increase the cache size. Default is 1024 KB.
Transaction Commit Control	Control to commit or roll back transactions based on the set of columns that pass through the transformation. Use the transaction commit control if you have a large amount of data and you want to control how it is processed. Does not apply when you select **Transformation Scope** as `Transaction` for real-time processing.
Transformation Scope	The method in which the Secure Agent applies the transformation logic to incoming data. Default is `All Input`. To process data from a real-time source, select the transformation scope as `Transaction`. For a source, which is not real-time, select the transformation scope as `All Input`. You can connect only one effective real-time source to a real-time target.

Input settings properties

You can enable **Sorted Input** under **Input Settings**. Sorted Input indicates that input data is presorted. Default is disabled.

Enable sorted input for better performance.

Note: When **Sorted Input** is enabled and the input is not sorted, the Secure Agent does not process input and the mapping fails.

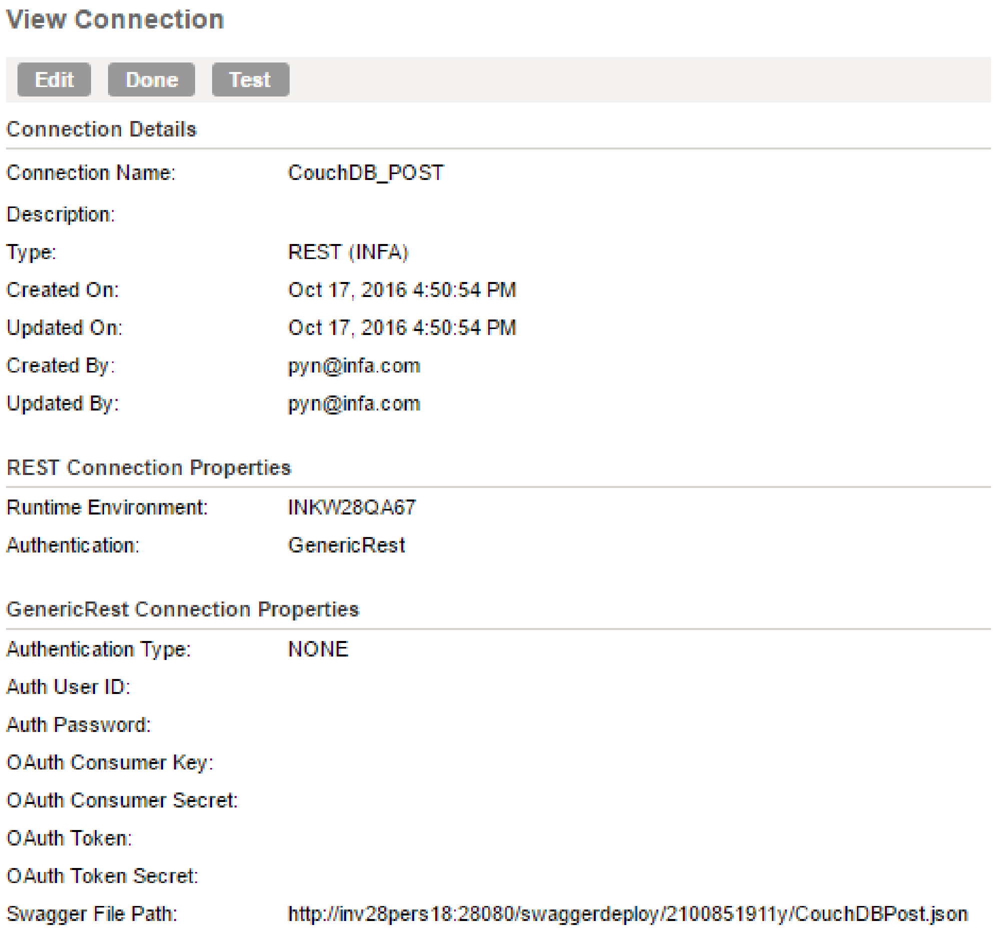
REST target mapping example

You are a human resources administrator and you want to update details of an employee in Apache CouchDB.

To update employee details in CouchDB from a flat file, perform the following tasks:

1. Create a flat file connection to read data from the flat file.
2. Create a REST connection to write data to CouchDB.

The following image shows the configured CouchDB connection:



Connection Details

Connection Name:	CouchDB_POST
Description:	
Type:	REST (INFA)
Created On:	Oct 17, 2016 4:50:54 PM
Updated On:	Oct 17, 2016 4:50:54 PM
Created By:	pyn@infa.com
Updated By:	pyn@infa.com

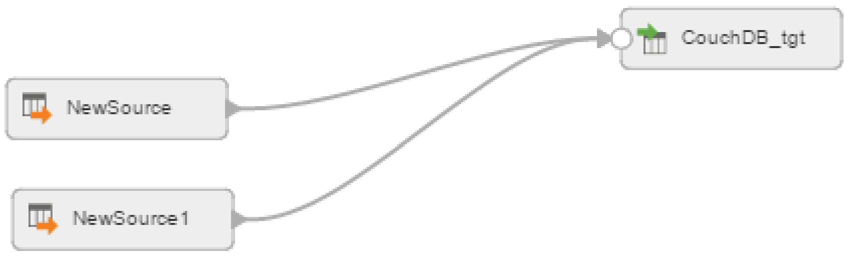
REST Connection Properties

Runtime Environment:	INKW28QA67
Authentication:	GenericRest

GenericRest Connection Properties

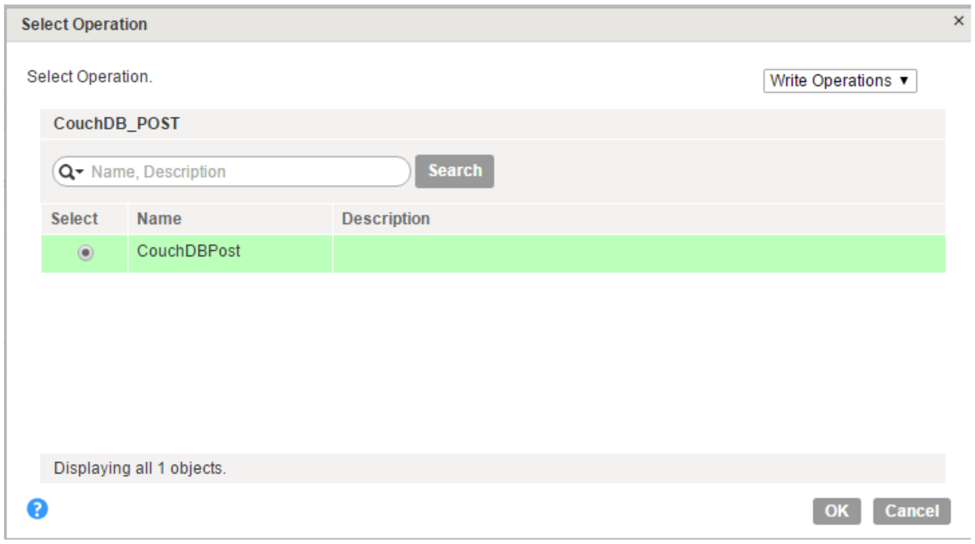
Authentication Type:	NONE
Auth User ID:	
Auth Password:	
OAuth Consumer Key:	
OAuth Consumer Secret:	
OAuth Token:	
OAuth Token Secret:	
Swagger File Path:	http://inv28pers18:28080/swaggerdeploy/2100851911y/CouchDBPost.json

3. Create a CouchDB mapping.



4. Add a Source transformation to include the flat file object that contains the employee details. Add the flat file connection.
5. Add a Target transformation to write the employee details to CouchDB. Perform the following tasks on the **Target** tab:
   a. In the **Connection** field, select the REST connection to connect to CouchDB.
   b. In the **Operation** field, select CouchDBPost as the operation.

The following image shows the CouchDBPost in the list of write operations:



Select Operation

Select Operation.

Write Operations ▼

CouchDB_POST

Name, Description | Search

Select	Name	Description
●	CouchDBPost	

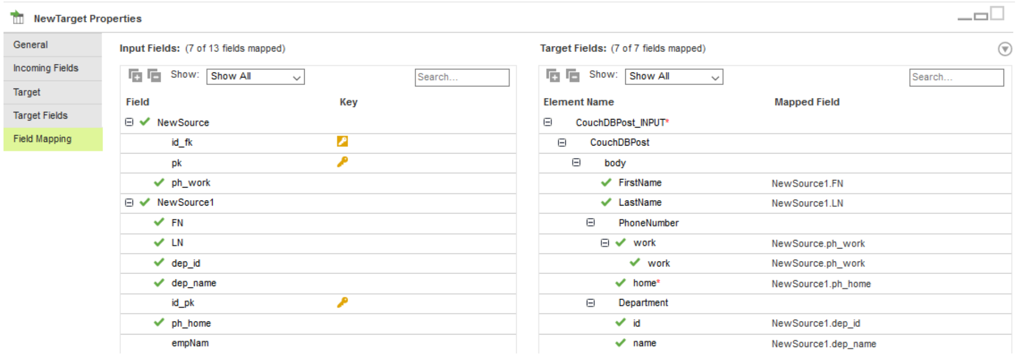
Displaying all 1 objects.

OK | Cancel

c. In the **Advanced Properties** section, set the cache size and the transaction commit interval.

6. On the **Field Mapping** tab, select the input elements to map to the target fields.

   The following image shows all the mapped fields between the input file and the CouchDB target:



NewTarget Properties

General | Incoming Fields | Target | Target Fields | Field Mapping

Input Fields: (7 of 13 fields mapped)

Show: Show All

Field	Key
⊟ ✓ NewSource	
id_fk	🔑
pk	🔑
✓ ph_work	
⊟ ✓ NewSource1	
✓ FN	
✓ LN	
✓ dep_id	
✓ dep_name	
id_pk	🔑
✓ ph_home	
empNam	

Target Fields: (7 of 7 fields mapped)

Show: Show All

Element Name	Mapped Field
⊟ CouchDBPost_INPUT*	
⊟ CouchDBPost	
⊟ body	
✓ FirstName	NewSource1.FN
✓ LastName	NewSource1.LN
⊟ PhoneNumber	
⊟ ✓ work	NewSource.ph_work
✓ work	NewSource.ph_work
✓ home*	NewSource1.ph_home
⊟ Department	
✓ id	NewSource1.dep_id
✓ name	NewSource1.dep_name

7. When you save and run the mapping in a mapping task, the Secure Agent updates the employee details in CouchDB.

CHAPTER 5

Swagger File Generation

Specify the path to a Swagger file that defines the REST service when you configure the connection. You can generate a Swagger file in Administrator.

Informatica Intelligent Cloud Services supports Swagger specification version 2.0. When you generate a Swagger file in Informatica Intelligent Cloud Services, you send an API call to the service using a request. If you do not have permissions to send an API call to the service, you can generate a Swagger file using a request and a sample response without submitting an API call.

You cannot modify a Swagger file once it is created. If you want to make changes in a Swagger file, create a new Swagger file.

Note: The swagger file generation functionality is available for the convenience of the REST V2 customers. Informatica does not warranty the compatibility of the swagger file for all customer scenarios.

Generating a Swagger File

You can generate a swagger file for REST V2 connections from the **Swagger Files** page in Administrator.

1. Click **New**.
2. Enter a name and description for the swagger file.
3. Specify the swagger details. The following table describes parameters to create a swagger file:

Parameter	Description
Runtime Environment	Mandatory. Name of the runtime environment used to generate the swagger file.
URL	Mandatory. URL consists of the host name and port number. For example: `http://localhost:8000`
Verb	Select the REST method being used by the web service. The supported methods are: - GET - POST - PUT - DELETE - PATCH

Parameter	Description
Authentication Type	If required, select the authentication method to login to the web service application. Default is none.
API Base Path	The path on which the API is served. The base path is the one specified after the hostname and port. For example, if the REST web service URL is `http://localhost:8000/greetings`, the base path will be `/greetings`.
API Path	The path specified after the base path is API path. For example, if the REST web service URL is `http://localhost:8000/greetings/hello`, the API path will be `/hello`. To define a path parameter, enclose the path, which is being treated as variable with curly brackets {}. For example, if the REST web service URL is `https://localhost:8080/sample/Stringoperation/concat/str1/str2?id=123` and concat is a variable in this path, define the API path as below: `Stringoperation/{concat}/str1/str2?id=123` You can define any number of path parameters. **Note:** The API path can include the query parameters. If you define the query parameters with the API path, do not specify the query parameters in the **Query Params** field.
Username	The user name to login to the web service application. Required for Basic and Digest authentication types.
Password	The password associated with the user name. Required for Basic and Digest authentication types.
Token	The access token to connect to the web service application. Required only for OAuth authentication type.
Token Secret	The password associated with the OAuth token. Required for OAuth authentication type.
Consumer Key	The client key associated with the web service application. Required for OAuth authentication type.
Consumer Secret	The client password to connect to the web service application. Required only for OAuth authentication type.
Accept	Select the MIME type.
Headers	Define header parameters in JSON format. For example: {"Accept-Charset":"utf-8"} Maximum length is 1020 characters.
Query Params	Provide query parameters in JSON format. For example: {"name":"subject","description":"The subject to be greeted."} Defining query parameters in the Query Params field adds the query parameters as input parameters in the Swagger specification file. If you define the query parameters in **Query Params**, do not specify the query parameters in the **API Path** field. Maximum length is 1020 characters.

Parameter	Description
Operation ID	Mandatory. A unique text identifier for the API path.
Content Type	Select the MIME type.

Raw Body	Enter the request body content. If you select `application/x-www-form-urlencoded` in the content type, specify the raw body parameters in the key-value pair. Each key-value pair starts with a new line. Example:
	`a : b`
	`c : d`
	`e : f`
	Not applicable to the GET method.

JSON Response File	Optional. Upload the JSON response file if you want to generate the swagger definition from the response file. No call is made to the REST endpoint if you select the JSON response file. If you do not provide the JSON response file, a call is made to the REST endpoint to fetch the response for generating swagger definition.

4. Click **Save** to generate the swagger file. An entry for the swagger file appears in the **Swagger Files** page.

   In case of failure while connecting to the web service, the fault response obtained from the web service is logged in the **Swagger Files** page.

5. Click the download icon to save the Swagger file in a local directory.

   To use the swagger file in the REST V2 connection, copy the file to the Secure Agent system where you will create a REST V2 connection.

APPENDIX A

Swagger objects

The following table lists swagger objects with field names honored by REST V2 Connector:

Object Category	Configurable Objects or Fields
Swagger	swagger info host basepath schemes consumes produces paths definitions parameters responses
Paths	/{path}
Path Item	$ref get put post delete parameters
Operation	Summary operationId consumes produces parameters responses schemes If operation ID value is not defined, the value of summary field is treated as operation ID.

Object Category	Configurable Objects or Fields
Parameter Type	path query header formData cookie body
Parameter Fields	name In. If `in` is `body`, the supported field is schema. If `in` is any value other than `body`, the supported fields are type, format, and items. required default. Not applicable to the `body` and `securityDefinitions` parameters.
Items	type format items
Response	schema
Schema	$ref format type items properties xml
Definitions	{name}
securityDefinitions	{name} You can configure Basic and apiKey security definitions types.
xml	name. Works only when used with the wrapped object. attribute prefix namespace wrapped

Important: Rest V2 Connector does not process objects and fields other than the listed above, in the swagger definition.

APPENDIX B

OpenAPI objects

The following table lists OpenAPI objects with field names honored by REST V2 Connector:

Object Category	Configurable Objects or Fields
openapi	openapi info servers paths components
Paths	/{path}
Path Item	operationId parameters requestBody responses
Parameter Type	path query header formData cookie
requestBody	description content
responses	status code such as 200 content
components	schemas requestBodies

Object Category	Configurable Objects or Fields
requestBodies	description content
Schema	$ref format type items properties xml

INDEX