Unica Link V12.1.2
REST Endpoint Configuration Guide

Contents

1 Introduction

The REST adapter uses a JSON configuration file to define endpoints for a service. The file is constructed by examining the documentation of a service and transcribing the details of the relevant endpoints into the configuration file.

2 Usage

Configuration files can be used in 2 ways:

- As a file in a HIP project that can be referenced from the REST adapter or REST node as a script.
- Installed into HIP as a packaged configuration. A configuration is a jar file containing the JSON configuration file, and optionally a Java plugin class and JSON templates describing requests and responses.

3 Format

The endpoint configuration file is a JSON file that contains some metadata about the service, and an array of endpoint definitions.

The top-level fields in the JSON are:

Field	Required	Type	Description
name	Y	string	A displayable name for the service
id	Y	string	An identifier for the service
description	Y	string	A displayable description for the service
authentication		object	An object that defines the authentication properties to be used when invoking APIs.
endpoints	Y	array of objects	The endpoint definitions for the service.
connection_test		string	The name of an endpoint which performs a connection test to the service.
properties		array of objects	***(Not currently used)***
schema_mapping		array of objects	Defines which endpoints are invoked to get field definitions for the service.
plugin		string	The classname for a Java plugin class if required to perform special processing.

3.1 Authentication

The authentication object defines the authentication scheme of the service. It has these fields:

Field	Required	Type	Description
type	Y	string	The authentication scheme. Permitted values are BASIC (Basic authentication), OAUTH2 (OAuth2 authentication) and USEENDPOINTDEF (each endpoint defines the authentication scheme, rather than defining at the service level)

username		string	The username for authentication connections in case of Basic authentication. For OAuth2, this is also required when accesstokenExpiryAction is set as "password" which means user credentials are required for getting new access token
password		string	Similar to username, password is required for authentication through Basic type and for OAuth2 when accesstokenExpiryAction is set to password.
tokenURL		string	For OAuth2, the URL for authentication provided by the service. This is invoked to acquire a new access token or for refreshing an expired token.
content_type		string	For OAuth2, the content type expected by the URL specified by tokenURL.
consumer_key		string	For OAuth2, the consumer key is used by the consumer to identify itself to the service provider. It is required when OAuth accesstokenExpiryAction is set to refresh or renew tokens.
consumer_secret		string	For OAuth2, the consumer secret is used to establish ownership of the consumer key. It is required when accesstokenExpiryAction is set to "password" or "client_credentials"
accesstokenExpiryAction		string	For OAuth2, specifies what action should be taken when the access token is invalid or expired. The permitted values for this property include: • **report_error** - report and error when the REST call to the service is unauthorized. • **refresh_token** – get an access token using the refresh token grant type implemented by the service. • **password** – get a new access token using username and password • **client_credentials** – request a new token using client_credentials grant type implemented by the service. This is a required property for OAuth2 authentication.
access_token		string	For OAuth 2.0, the initial access token.
security_token		string	For OAuth 2.0, set this if the service implementation requires a further token to be added to password for additional security.
refresh_token		string	For OAuth 2.0, if the accesstokenExpiryAction is set as refresh_token, this is required. It is usually provided to the user along with access_token when they have set up refresh token policy on the service side.
token_file		string	For OAuth2, when a new access token is retrieved it is saved in this specified file path.

encryption_key		string	For OAuth2, to keep the access token secure, it is encrypted using specified encryption_key.

3.2 Endpoints

The endpoint definitions define the APIs of the service. A definition has these fields:

Field	Required	Type	Description
name	Y	string	An identifying name for the endpoint. This name is referenced when invoking the endpoint
label	Y	string	A displayable label for the endpoint. This is shown in the user interface when listing the endpoints for a given service.
description	Y	string	A displayable description for the endpoint This is shown in the user interface when listing the endpoints for a given service.
method	Y	string	The HTTP method (also known as verb). Possible values are GET, POST, PUT, DELETE, HEAD and PATCH
url	Y	string	The URL of the API. This can contain property values specified as {$<property-name}
category		string	Used to categorize connection types.
authenticate		boolean	If true, authentication is required for the endpoint. If not specified, this defaults to false.
authentication		object	Defines the authentication to be used for the request.
request		object	Defines the attributes of the request. See the Request definition below.
responses		array of objects	Defines one or more possible responses for the endpoint. If the API can return different responses for different HTTP status codes (e.g. a successful response for 2xx statuses, and an error response for any other status code), then the request definition must list the statuses to which it applies. The format of a Response definition is given below.
enumeration		object	An enumeration definition is used to tie a property in a connector property definition to an endpoint that can be invoked to fetch enumerated values for the property. This is explained further below in Enumeration.
steps		array of strings	Steps provide a way to orchestrate one or more API calls together in a single endpoint definition. This is explained further in Steps.
retry		object	If retry object is specified, an endpoint may be invoked repeatedly. This is explained further below in Retry
assignments		array of objects	This provides a way to provide initial values for properties. See Assignments below.

3.2.1 Request

A request object defines properties of the request that are sent to the API. It has these fields:

Field	Required	Type	Description
template		string	The template references a JSON file that is included in the templates subdirectory of the configuration jar file. The template file is read when performing import when defining an action for a given endpoint.
content_type		string	The default content type of the data being sent in the request. If not set, defaults to application/json
parameters		List of objects	Defines query parameters for a request. This section is optional: query parameters can also be specified in the URL string. Parameters can also be used to build requests by setting the location to "BODY". See Parameters below.
headers		List of objects	Defines header values for the request. The value of headers can be passed as properties. Each header is defined as a Parameters object which is defined below.
is_formdata		Boolean	Set to true if the request requires form data.
formdata_parts		List of objects	Defines the list of form data parts if is_formdata is set to true. See section Form Data for an explanation as to how form data is treated by the REST adapter.
body_from_template		Boolean	If true, the request body will be constructed from the template provided in the 'template' field.

3.2.2 Response

A response object defines properties of the response that are returned from an API. It has these fields:

Field	Required	Type	Description
status		integer	An HTTP status code to which the reply is applicable. This is optional, but if more than one response is listed, then each response must have a status code. [This ought to be an array of status codes, or allow ranges or lists, e.g. 4xx]
template		string	The template references a JSON file that is included in the templates subdirectory of the configuration jar file. The template file is read when performing import when defining an action for a given endpoint.
assignments		List of objects	Assign fields from the response to the property set. This provides a convenient way to pass values between REST calls that are invoked via steps. See Assignments section below.
filter		object	A filter provides a means to filter what is returned from the response. Often a response format has many fields which are of no interest. See Filter section below.
conversion		string	Specify a conversion function to convert the response to another format. The only supported value is " XML2JSON" which results in an XML response being converted to its corresponding JSON form.

3.2.3 Enumeration

The enumeration element identifies an endpoint to invoke to return a set of enumerated values for a given property. The name of the enumeration must match the enumeration field in the corresponding property in the connector descriptor. For example, a connection type for a given service may have a property named template defined as follows:

```
{
    "label": "Email Template",
    "name": "template",
    "type": "string",
    "required": true,
    "enumeration": "templates",
    "description": "The template to use for the campaign",
    "scope": "target_action"
}
```

In the corresponding configuration file, there must be an endpoint which has an enumeration object with a matching name (e.g. "templates"). The user interface for the connection type will then invoke this endpoint to enumerate the list of values.

The enumerate object has the following fields:

Field	Required	Type	Description
name	Y	string	The name of the enumeration
array_path	Y	string	The path of the JSON array containing the repeating elements. If the response is a JSON array the path should be specified as "/".
value_path	Y	string	The relative path within the returned object containing the enumerated value.
label_path		string	If the returned object has a field that provides a descriptive name, this should be set to that relative path.
qualifier_path		string	If the object requires an additional qualifying field, then this path specifies that qualifier. The qualifier is added to the label in the user interface. Qualifiers may be necessary to differentiate objects when the labels are not necessarily unique

For example, if the response JSON has this content:

```
{
    "templates": [
        {
            "id": 123,
            "name": "My template",
            "data": "abc"
        },
        {
            "id": 678,
            "name": "My other template",
            "data": "def"
        }
    ]
```

```
}
```

The enumeration object for this response would have this content:

```
{
    . . .
    "enumeration": {
        "name": "templates",
        "array_path": "templates",
        "value_path": "id",
        "label_path": "name",
        "qualifier_path": "id"
    }
}
```

This would result in these enumeration values being displayed in the UI:

Value	Label
"123"	My template (123)
"678"	My other template (678)

3.2.4 Steps

Steps provide a way to invoke multiple APIs in sequence. Typical use cases are:

- A logon API must be invoked to get a token that is then used in a header in a subsequent call
- An API starts a task and returns an ID, and another API is called to get the status of the task, passing the ID as a query parameter.

The steps element is an array of endpoint names. In some cases, an endpoint definition may contain no content other than steps. For example,

```
{
    "name": "getMergeFields",
    "label": "Get Fields For Campaign",
    "steps": [
        "getCampaign",
        "getMergeFields"
    ]
}
```

When invoked, this endpoint will first call the getCampaign endpoint, which would contain an assignment element to get some contextual information from the response (e.g. an id field is assigned to property "campaign_id"). If the first step is successful, then the second endpoint (getMergeFields) is invoked. The URL, header or request for this endpoint would reference the property set by the first call, for example "/server/campaigns/{$campaign_id}"

3.2.5 Retry

If specified, the retry element specifies properties that determine conditions for retrying the endpoint. This can provide a way to poll an endpoint until some condition is satisfied. The fields of a retry definition are:

Field	Required	Type	Description
interval	Y	integer	The interval between calls, specified in milliseconds.
attempts	Y	integer	The number of times to make the call before stopping.
retry_on_error		boolean	If true, the API will be retried if it does not return a 2XX success status.
condition		string	The condition to check to determine whether another call should be made. The permitted expression formats are detailed below.

The condition can be any of these expressions:

Expression:

```
{$<property>} <operator> <value>
```

Property – the name of a property

Operator – one of:

Operator	Meaning
==	Equal to
!=	Not equal to
IN	The value is one of the supplied values
!IN	The value is not one of the supplied values
<	For integer properties, less than
>	For integer properties, greater than
<=	For integer properties, less than or equal to
>=	For integer properties, greater than or equal to

Value – The value to compare the property against. If the operator is IN or !IN, then the value should be a comma delimited set of values enclosed in square brackets with no spaces. e.g. [val1,val2,val3]

Example:

```
{$status} != "complete"
```

Simple Expression:

```
<expression> <logical operator> <expression>
```

Expressions are expressed as above.

The logical operator can be either 'OR' or 'AND'.

There can only be 2 expressions and a single logical operator.

Example:

```
{$status} != "complete" AND {$status} != "error"
```

Note that this can be also expressed using the IN operator:

```
{$status} !IN ["complete","error"]
```

Complex Expression:

```
(<simple expression>) <logical operator> (<simple expression>)
```

Simple expressions are expressed as above.

The logical operator can be either 'OR' or 'AND'.

There can only be 2 simple expressions and a single logical operator.

Example:

```
({$P1} == 10 AND {$P2} == 20) OR ({$P3} == 30 AND {$P4} == 40)
```

An example of a retry expression in an endpoint definition is:

```
{
    "name": "batchStatus",
    . . .
    "responses": [
        {
            . . .
            "assignments": [
                {
                    "name": "batch_status",
                    "location": "BODY",
                    "path": "status"
                }
            ]
        }
    ],
    "retry": {
        "interval": 5000,
        "attempts": 100,
        "condition": "{$batch_status} != \"finished\""
    }
}
```

In this example, the property batch_status is set to the status field in the response. If the value of batch_status is not the string "finished", then the call will be repeated every 5 seconds, for up to 100 iterations.

3.2.6 Parameters

Parameter objects are used to define query parameters, headers and formdata parts:

Field	Required	Type	Description
name	Y	string	The name of the parameter
value	Y	string	The value for the parameter. Commonly, this will be set to a property using the {$<property>} notation.
is_required		boolean	If set to true and the value is not provided an error will be raised.
location		string	Location can be set to either "QUERY" or "BODY". If "QUERY", the parameter is defining a query parameter. If "BODY", the parameter is defining a field within the request body. In this case the "path" must also be set to provide the path of the JSON field within the request.
path		string	If location is "BODY" the path must be set to provide the path to the JSON field to set in the request.

3.2.7 Assignments

Assignments provide a mechanism to assign the value of a property to some element of the response. This is necessary when using steps within an endpoint, and some data elements need to be passed from one step to another. For example, a logon API may return a token that is then set in a header in a subsequent API call. The fields of an assignment are:

Field	Required	Type	Description
name	Y	string	The name of the property to which the value is assigned
value		string	The literal value to set the property to. One of value or location must be specified.
location		string	This can either be set to BODY or HEADERS. If BODY, then path defines the path of the JSON field within the response, e.g. "address.city". If HEADERS, then the path specifies the name of the header field. One of value or location must be specified.
path		string	The path of the JSON field, or the name of a header, depending on the value of location.

Assignments can also be set in an endpoint definition. These assignments will be made before the endpoint is invoked, providing a way to provide default values for properties.

3.2.8 Filter

The filter object specifies JSON fields to include or exclude in the response data.

Field	Required	Type	Description
type	Y	string	The type of filter to apply. This must be either "INCLUDE", "EXCLUDE" or "EXCLUDE_ARRAY_ELEMENTS". If set to INCLUDE, then only the specified paths will be returned. If set to EXCLUDE, then all paths will be returned other than the listed paths. EXCLUDE_ARRAY_ELEMENTS provides a way to remove certain elements from a JSON array.

paths	Y	array of strings	A list of paths to either include or exclude. Paths use dot notation. For example, if a JSON response contains an address field which is a JSON object containing a state field, "address" identifies the address object, whereas "address.state" identifies the state field within the address object.
array_path		string	When type is EXCLUDE_ARRAY_ELEMENTS, the path to the array in the response.
values		array of strings	When type is EXCLUDE_ARRAY_ELEMENTS, a list of paths relative to the array that gathers values to test in the condition
condition		string	When type is EXCLUDE_ARRAY_ELEMENTS, a condition to determine whether elements are included or excluded.

An example of use of EXCLUDE_ARRAY_ELEMENTS is:

```
{
    "type": "EXCLUDE_ARRAY_ELEMENTS",
    "array_path": "clicks_detail",
    "values": [ "ts" ],
    "condition": "{$since_date} > {$ts}"
}
```

This filter will remove array elements from the array at path "clicks_detail" where the value of "clicks_detail.ts" is less than the property "since_date".

3.3 Properties

Properties are currently not used. The intention of the properties section is to define properties that are used within the endpoint definitions.

3.4 Schema Mapping

The schema mapping section is optional, but is required for connection types deployed to HCL Link. For a given condition, it defines a set of field definitions that should be returned when that condition is true. A condition is a property value having a certain value. The set of field definitions can be composed of static and dynamic elements. Static field definitions define fields that are always returned. Dynamic elements specify an endpoint that should be called to get field definitions, and additional attributes that define how to interpret the results of the endpoint.

The schema_mapping is an array of objects where each object has these fields:

Field	Required	Type	Description
name		string	A descriptive name for the mapping definition
condition		string	A condition which if true selects this mapping definition. The condition is required if there is more than one mapping but should not be specified if there is only one.
static		array of objects	One of more statically defined fields
dynamic		array of objects	One or more dynamically defined fields

format		string	The type of generated schema format. This can have the values "json" or "tree".
mode		string	Can have the values "request", "reply" or "both". Generated schema can be different for request and response mode. If not specified, or specified as "both", the schema mapping is valid for both request and response types.
jsonSchema		object	When the format is "json", this field provides a fixed JSON format that is returned as the schema.

A static definition has these fields

Field	Required	Type	Description
internal_name	Y	string	The name of the field
external_name	Y	string	The descriptive name of the field suitable for displaying in a UI
description		string	A description for the field which provides an explanation of the purpose of the field
type	Y	string	The type of the field. It must be one of: **text** – a textual field **integer** – an integer field **number** – any numeric field (integer or decimal) **table** – specifies that there are child field definitions. This is only applicable to dynamic definitions.
default		string	A default value for the field.
required		boolean	Whether the field is required or not. Defaults to false.

A dynamic definition has these fields:

Field	Required	Type	Description
endpoint	Y	string	The name of the endpoint to invoke to fetch the dynamic fields
paths	Y	object	The paths of the dynamic field attributes in the JSON response returned from the endpoint.
type_mapping		array of objects	A list of mappings from types of the endpoint's response to the internal types listed above under the type attribute of static definitions.
schema_path		string	When jsonSchema is configured, the generated dynamic fields will be set in the specified schema_path of static JSON.

The path object for a dynamic definition has these fields:

Field	Required	Type	Description
array	Y	string	The path of the array containing the field definition objects

child_array		string	The path of the array containing child field definitions relative to the array object. This is only applicable for a table type property.
internal_name	Y	string	The path of the field name
external_name		string	The path of the external displayable name of the field. If not specified, the external name will be set to the internal name.
default		string	The path of the default value of the field
description		string	The path of the description for the field
required		string	The path of the required attribute for the field. The value can be a boolean (true/false), or a string value("true"/"false")
type		string	The path of the field type. If not specified, the type of the field will be set to "string"

The type mapping object has these fields:

Field	Required	Type	Description
from	Y	string	The type returned in the response
to	Y	string	The type to which the type should map. The permitted values are shown above in the static definition object.

3.4.1 Schema Mapping Example

Consider the following schema mapping definition:

```
{
    "schema_mapping": [
        {
            "name": "Get merge fields",
            "condition": "{$operation} == \"new\"",
            "static": [
                {
                    "internal_name": "email",
                    "external_name": "Email",
                    "description": "Email address",
                    "type": "text",
                    "default": null,
                    "required": true
                }
            ],
            "dynamic": [
                {
                    "endpoint": "getMergeFields",
                    "paths": {
                        "array": "merge_fields",
                        "external_name": "name",
                        "internal_name": "tag",
                        "default": "default_value",
                        "required": "required",
                        "type": "type"
                    },
                    "type_mapping": [
```

```
                        {
                            "from": "int",
                            "to": "integer"
                        },
                        {
                            "from": "text",
                            "to": "string"
                        }
                    ]
                }
            ]
        }
    ]
}
```

An explanation of this JSON:

- The condition specifies that this schema mapping definition is applicable when the property operation has the value "new". Since there is only one schema mapping object in the array, the condition should not be specified, but is shown here simply to provide an example.
- This schema is defined by a single static field + the fields returned from the dynamic endpoint getMergeFields
- The static field defines an email field
- The dynamic definition specifies that the getMergeFields should be invoked to get the field definitions.
    - The paths object specifies the paths of the JSON fields providing the field definition
    - The types array specifies the mapping of the contents of the type field to the internal types.

The paths in this object corresponds to this JSON response:

```
{
    "merge_fields": [
        {
            "name": "field1",
            "tag": "Field1",
            "default_value": "100",
            "required": true,
            "type": "int"
        }
    ]
}
```

4 Paths

Paths are used in some elements to identify fields in JSON objects. Paths are expressed using dot notation. For example, consider the JSON:

```
{
    "a": 1,
    "b": {
       "c": {
          "d": 2,
          "e": [ 10,11,12 ]
```

```
        }
    }
}
```

The paths for the fields in this JSON are:

- a
- b
- b.c
- b.c.d
- b.c.e

5 Authentication

For any API call, authentication can be defined. If all APIs use the same authentication scheme, then this can be defined in the top-level of the configuration. If certain endpoints require different schemes, then each endpoint can have its own endpoint definition. If an endpoint definition defines authentication, then that definition is used, otherwise the top-level definition is used.

Authentication is only applied to an API call if the "authenticate" attribute for the endpoint is set to true.

6 Form Data

Form data can be sent in requests in 2 ways:

- Specify the form fields in the configuration file
- Pass JSON containing the form fields.

In both cases, the request must have "is_formdata" set to true.

6.1 Fields defined in Configuration File

This approach provides a way to build the request from property values. Request data need not be passed to the REST adapter or node.

To specify the form fields in the configuration file, the "request" element contains a "formdata_parts" element which is a list of Parameters objects. For example,

```
{
    "request": {
        "is_formdata": true,
        "formdata_parts": [
            {
                "name": "FirstName",
                "value": "{$first_name}",
                "is_required": true
            },
            {
                "name": "LastName",
                "value": "{$last_name}",
                "is_required": true
            },
            {
                "name": "Age",
                "value": "{$age}"
```

```
            }
        ]
    }
}
```

The form data request will be constructed from the elements specified in the formdata_parts array. The values can either be literal values or properties.

6.2 Fields passed in JSON object

An alternate approach is to pass the form data as a JSON object where the JSON fields are the form data field names. Using the same example shown above, the JSON object passed as a request would be:

```
{
    "FirstName": "Fred",
    "LastName": "Quimby",
    "Age": 90
}
```

If the endpoint allows multiple fields with the same name, arrays of values can be passed for a given field. For example, if a service allowed for multiple phone numbers to be passed as form data field "TelNum", the request would be:

```
{
    "FirstName": "Fred",
    "LastName": "Quimby",
    "Age": 90,
    "TelNum": [ "+11115551111", "+12225552222", "+13335553333" ]
}
```