



Integrating SailPoint with Web Services Connector

Version: 8.1 Patch 4

Copyright and Trademark Notices

Copyright © 2021 SailPoint Technologies, Inc. All Rights Reserved.

All logos, text, content, including underlying HTML code, designs, and graphics used and/or depicted on these written materials or in this Internet website are protected under United States and international copyright and trademark laws and treaties, and may not be used or reproduced without the prior express written permission of SailPoint Technologies, Inc.

"SailPoint," "SailPoint & Design," "SailPoint Technologies & Design," "Identity Cube," "Identity IQ," "IdentityAI," "IdentityNow," "SailPoint Predictive Identity" and "SecurityIQ" are registered trademarks of SailPoint Technologies, Inc. None of the foregoing marks may be used without the prior express written permission of SailPoint Technologies, Inc. All other trademarks shown herein are owned by the respective companies or persons indicated.

SailPoint Technologies, Inc. makes no warranty of any kind with regard to this manual or the information included therein, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. SailPoint Technologies shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Patents Notice. <https://www.sailpoint.com/patents>

Restricted Rights Legend. All rights are reserved. No part of this document may be published, distributed, reproduced, publicly displayed, used to create derivative works, or translated to another language, without the prior written consent of SailPoint Technologies. The information contained in this document is subject to change without notice.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c)(1) and (c)(2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

Regulatory/Export Compliance. The export and re-export of this software is controlled for export purposes by the U.S. Government. By accepting this software and/or documentation, licensee agrees to comply with all U.S. and foreign export laws and regulations as they relate to software and related documentation. Licensee will not export or re-export outside the United States software or documentation, whether directly or indirectly, to any Prohibited Party and will not cause, approve or otherwise intentionally facilitate others in so doing. A Prohibited Party includes: a party in a U.S. embargoed country or country the United States has named as a supporter of international terrorism; a party involved in proliferation; a party identified by the U.S. Government as a Denied Party; a party named on the U.S. Department of Commerce's Entity List in Supplement No. 4 to 15 C.F.R. § 744; a party prohibited from participation in export or re-export transactions by a U.S. Government General Order; a party listed by the U.S. Government's Office of Foreign Assets Control as ineligible to participate in transactions subject to U.S. jurisdiction; or any party that licensee knows or has reason to know has violated or plans to violate U.S. or foreign export laws or regulations. Licensee shall ensure that each of its software users complies with U.S. and foreign export laws and regulations as they relate to software and related documentation.

Contents

Integrating SailPoint with Web Services	1
Supported Features	1
Supported Managed Systems	2
Prerequisites	2
Required Permissions	2
Upgrade Considerations	2
Connecting SailPoint and Web Services	4
Configuring the Connector in SailPoint	4
Configuration Parameters	4
Configuring Web Services for Integration	37
Pass Through Authentication	37
Multiple Independent Endpoints	38
Pagination	39
Enabling Single API Calls	47
Partitioning	48
Configuring Multiple Entitlement Requests	52
Schema Attributes	54
Web Services Before/After Operation Rule	55
WebServicesBeforeOperationRule	55
WebServicesAfterOperationRule	58
Web Services Class used in Before/After Operation Rule	59
Troubleshooting	67

Integrating SailPoint with Web Services

Connecting SailPoint to your Web Services allows you to configure any web service supported managed system which can read and write on the managed system using the respective managed system Web Services.

Web Services supports JSON and XML for read and write.

Supported Features

The Web Services Connector supports the following features:

Account Management

- Aggregation, Refresh Accounts, Pass Through Authentication (Basic Authentication)
Web Service Connector provides support for using Web Service application as a Pass Through Authentication application. For more information on configuration for pass-through authentication, see [Pass Through Authentication](#).

Currently Pass Through Authentication is supported with identity attribute only.

- Create, Update, Delete
- Enable, Disable, Change Password
- Add /Remove Entitlements

Account - Group Management

- Aggregation, Refresh Groups

Additional Supported Features

SailPoint Web Services Connector now:

- Application layer proxy
- Provides support for saving of updated **Refresh Token** received along with access token, if any.
If **Refresh Token** has expired, it must be manually generated and updated in the application configuration as mentioned in [\(General Settings\) Basic Configuration Parameters](#).
- Provides additional support for client certificate authentication. For more information, see “Enable Client Certificate Authentication” parameter in [\(General Settings\) Basic Configuration Parameters](#).
- Supports account, group and role aggregation from multiple independent endpoints. For more information, see [Multiple Independent Endpoints](#).
- Provides support of cookies for multiple endpoints configuration. The application will manage the cookies internally only for the multiple endpoints configured for the same operation. Cookies from the previous endpoints can be used in all the subsequent endpoints of same operation type.
- Provides additional support for pagination.

For more information on embedding pagination support in Web Service Connector, see [Pagination](#).

- Provides support for partitioned account aggregation and configuration for single-threaded aggregation and partitioned aggregation simultaneously.

For more information, see [Partitioning](#).

- Support for Multiple Group

The following table lists the example for different operations for the added new Group Object Types:

Object Types	Operation Type	Description
Group	Group Aggregation	Aggregates all Group objects.
	Get Object - Group	
	Add Entitlement	
	Remove Entitlement	
Role	Group Aggregation - Role	Aggregates all Group Role objects.
	Get Object - Role	
	Add Entitlement - Role	
	Remove Entitlement - Role	
PermissionSet	Group Aggregation - PermissionSet	Aggregates all Group PermissionSet objects.
	Get Object - PermissionSet	
	Add Entitlement - PermissionSet	
	Remove Entitlement - PermissionSet	

Supported Managed Systems

SailPoint Web Services supports web services with JSON/ XML response format.

Prerequisites

Web Services must be accessible.

Required Permissions

The user /administrator must have the required permissions to call the web services API of the managed system.

Upgrade Considerations

- After upgrading IdentityIQ from version 7.2 Patch 3 or earlier to version 8.0 Patch 5 or higher version, add the following entry key in the application debug page of the existing application:

```
<entry key="encrypted" value="accesstoken,refresh_token,oauth_token_
info,client_secret,private_key,private_key_
password,clientCertificate,clientKeySpec"/>
```

- After upgrading IdentityIQ from version 7.2 Patch 3 or earlier to version 8.0 Patch 5 or higher version:
 - to support pass-through authentication, add **isGetObjectRequiredForPTA** attribute to the application debug page.
For more information on the above additional configuration attribute, see [Additional Configuration Parameters](#).
 - if the **Authentication Method** is selected as **OAuth2** and the **Grant Type** as **JWT** then add the following parameters in the application debug page:
 - **oAuthJwtHeader**
 - **oAuthJwtPayload**

For more information on the above additional configuration attributes, see [Additional Configuration Parameters](#) section.

- Add the following attribute in the featureString in the application debug page:

AUTHENTICATE

For more information on the above attribute, see [Keywords](#) section.

- To enable unlock feature, add the **UNLOCK** feature value to the **featuresString** in application debug page as follows:

```
featuresString="UNLOCK"
```

- All existing applications would work seamlessly. However, the multiple independent endpoints for aggregation and get object operations would only be supported for new applications.
- Endpoints must have non-empty unique names.
- Use of quotes is not applicable. Hence for applications created before IdentityIQ version 8.0 Patch 5 and require use of quotes, contact **SailPoint Customer Support**.
- All existing applications will work seamlessly. To use the new feature of Partitioning for existing application (from version 8.0 Patch 5 onwards) or new application user must add static mappings to get Partition objects or configure Dynamic / Get Partition endpoint. To aggregate accounts using partitions, user must configure Partitioned Account Aggregation endpoint. For more information, see [Partitioning](#).

This feature is not supported for the application which are created before IdentityIQ version 8.0 Patch 5.

Connecting SailPoint and Web Services

To connect SailPoint and Web Services, perform the following tasks:

- [Configuring the Connector in SailPoint](#)
- [Configuration Parameters](#)

Configuring the Connector in SailPoint

Use the **Edit Application** page to define the Web Services Connector application in your enterprise.

This procedure provides the basic information necessary to connect your connector. For additional information, see the [IdentityIQ Application Configuration Guide](#).

1. Navigate to **Applications > Applications Definitions**.
The Edit Application page appears.
2. Enter the following information on the **Edit Application** page:
 - **Name** - The name of the application. This is the name used to identify the application throughout IdentityIQ.
 - **Owner** - The owner of the application. The owner specified here is responsible for certifications and account group certifications requested on this application if no revoker is specified.

Application ownership can be assigned to an individual identity or a workgroup. If the application ownership is assigned to a workgroup, all members share certification responsibilities, are assigned certification request associated with the application, and all can take action on those requests.
 - **Application Type** - The **Application Type** drop-down list contains the types of application to which IdentityIQ can connect. This list will grow and change to meet the needs of IdentityIQ users.
3. Select the **Configuration > Settings** tabs and enter the information required for IdentityIQ to connect and interact with the application. The information required varies by application.
4. Click **Save**.
The Edit Application <application> page appears.
5. Click **Test Connection** to verify the information is correct.

Configuration Parameters

This section contains the information that the connector uses to connect and interact with Web Services system through the application. Each application type requires different information to create and maintain a connection.

All the attributes marked with * are mandatory attributes.

SailPoint IdentityIQ Web Services Connector provides the following type of configuration parameters:

- [\(General Settings\) Basic Configuration Parameters](#)
- [Operation Specific Configuration Parameters](#)

- [Additional Configuration Parameters](#)
- [Application Layer Proxy Support](#)

(General Settings) Basic Configuration Parameters

The parameters and their respective descriptions of the parameters are mentioned in the following format:

Parameter Name

Description of the parameter.

Following are the list of basic configuration parameters of SailPoint IdentityIQ Web Services Connector:

Add Object Type

This button pops up a window to add the name of the object type. For example, **Group Aggregation - Role**

Base URL *

The base URL to connect to the web service managed system.

Authentication Method*

Authentication method that is supported by the managed system

- *OAuth2*
- *API Token*
- *Basic Authentication*
- *No/Custom Authentication*

If **No/Custom Authentication** is selected, for more information related to specific use case configuration, see [Configuration for No/Custom Authentication](#).

SOAP Web Services supports only 'Basic Authentication' method.

Schema Attribute for Account Enable status

Attribute name and value required to be provided to check the Enable status.

For example: `status=Active`

This attribute supports only single value as described in the above example and does not support the conditional operators as follows:

```
"status=Active || status=Pending"
```

```
"status=Active && status=Pending"
```

Request Timeout (In Seconds)

Request Timeout Value in seconds.

Enable Client Certificate Authentication

Configure client certificate authentication.

- **Client Certificate***: Client certificate for authentication.
- **Certificate Key***: Client certificate's private key.

Web Services Connector supports only PEM format for the 'Client Certificate' and certificate's private key.

Attributes for Authentication Method: OAuth2

Grant Type*

Select the type of Grant:

- Refresh Token
- JWT
- Client Credentials
- Password
- SAML Bearer Assertion

Client Id*

(Optional for JWT and SAML Bearer Assertion) Client Id for OAuth2 authentication.

Client Secret*

(Optional for JWT and SAML Bearer Assertion) Client Secret for OAuth2 authentication.

Token URL*

URL for generating access token.

Token URL supports placeholders for replacement of application attribute dynamically.

For example, if token URL is required to be prepared with some sensitive information that is, `client_secret`, then `$application.client_secret$` can be used so that the corresponding value would be determined from the application replaced at the appropriate location in the token URL.

SAML Assertion URL *

(Applicable if Grant Type is selected as SAML Bearer Assertion) URL for generating SAML Assertion.

For example, if SAML Assertion URL is required to be prepared with information such as password, then `$application.saml_password$` can be used so that the corresponding value would be determined from the application and replaced at the appropriate location in the SAML Assertion URL.

SAML Request Body *

(Applicable if Grant Type is selected as SAML Bearer Assertion) Request Body for generating SAML Assertion.

SailPoint recommends to use `$application.saml_username$` and `$application.saml_password$` placeholder in request body to hide sensitive data.

Username*

(Applicable if Grant Type is selected as **Password** and optional for **SAML Bearer Assertion**) Username of the resource owner.

Password*

(Applicable if Grant Type is selected as **Password** and optional for **SAML Bearer Assertion**) Password of the resource owner.

Refresh Token*

(Applicable if Grant Type is selected as Refresh Token) A valid refresh token for grant type authentication.

Private Key*

(Applicable if Grant Type is selected as JWT) The private key to be used to sign the JWT.

Private Key Password*

(Applicable if Grant Type is selected as JWT) Password for the provided private key.

Attribute for Authentication Method: API Token

API Token*

Enter the API token specific to the Managed System.

Attributes for Authentication Method: Basic

Username*

Username that holds permission to execute the Web Service.

Password*

Password of the user name.

For more information on the additional configuration attributes for OAuth 2.0 authentication method, see [Additional Configuration Parameters for OAuth2](#).

Configuration for No/Custom Authentication

When **Authentication Method** is selected as **No/Custom Authentication** it means that Web Services would fetch the authentication details for all other endpoints.

Perform the following configurations:

1. Select the **Authentication Method** as **No/Custom Authentication**.

If an endpoint does not require authentication, you only need to select **No/Custom Authentication**. Continue to the following steps to set up custom authentication

2. Select the endpoint operation type as **Custom Authentication**.

3. Perform the following for endpoint configuration:

- **Authentication URL:** From the endpoint details page, enter the full URL in the **Authentication URL** field.

Ensure that you enter the full URL in this field unlike the other endpoints where this field takes in the context URL)

- **Header and Body:**

- Configure Header and Body with the required fields for the token generation.
- Sensitive attributes in Header and Body must not be passed through UI. They can be added through the application debug page. While adding sensitive attributes ensure that the attributes are added with a suffix **_CA**.

For example, to use password in the request add **password_CA** in the debug page.

If the attributes are not suffixed with **_CA** Web Services might display unusual behavior.

- The keys for sensitive attributes updated above must be appended in the encrypted list in the application.

For example if the attribute updated is password then it must be added as follows:

Before updating the encrypted list:

```
<entry key="encrypted" value="accesstoken,password,refresh_token,oauth_token_info,client_secret,private_key,private_key_password,clientCertificate,clientKeySpec,resourceOwnerPassword,custom_auth_token_info"/>
```

After updating the encrypted list:

```
<entry key="encrypted" value="accesstoken,password,refresh_token,oauth_token_info,client_secret,private_key,private_key_password,clientCertificate,clientKeySpec,resourceOwnerPassword,custom_auth_token_info,password_CA"/>
```

- **Response Mapping:** Configure **Response Attribute Mapping** to obtain the access details from the response. Response mapping would be same as of the other endpoints where key would be added under **Response Attribute** and its respective path must be added under **Attribute Path**.

The value under **Response Attribute** must not be present in the schema. After response parsing the value would be stored in key value format as follows:

```
<entry key="accesstoken" value="access_token"/>
```

- The token value generated after executing **Custom Authentication** endpoint can be utilized in all other endpoints using placeholders as follows:

```
"$application.<your key>$"
```

For example:

```
"$application.accesstoken$".
```

If token type is for **Bearer** instance, then it can be used as "**<your token type> \$application.<your key>\$**"

For example:

```
"Bearer $application.accesstoken$"
```

Operation Specific Configurations

No default provisioning template is provided. The template may vary from one managed system to another.

Connector Operations

Perform the following procedure to add and configure the specific operations:

1. Click **Add Operation**.
2. Select the operation from the drop down list of **Choose Operation**.
3. Provide a unique name to the operation. For example: **Account Aggregation-1, Get Object-Role, Group Aggregation-Role**.
4. Select the configure option (Pencil image) on the same row and configure the newly added operation. For more information on the operation specific configuration parameters, see [Operation Specific Configurations](#) which allows user to provide additional options.

For certain operations, the Body must be updated accordingly as mentioned in the [Supported Operations](#).

Operation Specific Configuration Parameters

Following are the list of operation specific configuration parameters of SailPoint Web Services:

Enable cURL

Checkbox to enable configuration using cURL command.

User can configure an operation using **cURL Command** or through the Context URL, Method, Header and Body. While configuring an operation through **cURL Command**, the **Context URL**, **Method**, **Header** and **Body** operation specific parameters would not be accessible.

For more information, see the **cURL Command** explained below.

cURL Command

The endpoint cURL command for operation which contains the complete request URL, Method, Header and Body as provided in the following example:

```
curl --location --request POST 'https://api.dropbox.com/2/team/members/list' \  
--header 'Authorization: $application.accesstoken$' \  
--header 'Content-Type: application/json' \  
--data-raw '{"limit":1}'
```

In the above example,

- complete request URL: `https://api.dropbox.com/2/team/members/list`
- Method: `POST`
`--header 'Authorization: $application.accesstoken$' \
--header 'Content-Type: application/json' \`
- Header
- Body: `--data-raw '{"limit":1}'`

Context URL

Context URL specific to the operation.

For example:

```
/api/core/v3/securityGroups?startIndex=0&count=100&fields=%40all&sort=lastNameAsc
```

Following are the keywords that would be used in the Context URL along with the placeholder \$:

- plan
- response
- application
- getobject
- authenticate

For more information on the Keywords, see [Keywords](#).

Method

Select one of the following type of HTTP method supported by the respective operation:

- GET
- PUT
- POST
- DELETE
- PATCH

Header

(*Optional*) To view the header value in plain text, user must provide it in encrypted form. The encrypted value can be obtained from IdentityIQ Console.

For example: The following example displays the sample header key and header value, where **Authorization** is header key and **1:vQaPY5LvJVbpsaig0nE56Q==** is the header value:

```
Authorization 1:vQaPY5LvJVbpsaig0nE56Q==
```

Content-Type header value must contain type matching any XML formats that is, application/XML or text/XML or */XML.

JSON request, JSON response: Content-Type= application/JSON (*optional*), Accept (*optional*)

XML request, XML response: Content-Type= application/XML or text/XML or */XML (*required*), Accept (*optional*)

JSON request, XML response: Content-Type=application/JSON (*optional*), Accept= application/XML or text/XML or */XML (*required*)

Body

Standard http body used to post data with request. User can send data in either of the following format:

form-data: (*Applicable only for JSON*) Key value. User must set the data that has to pass in the key value

raw: Data to be sent in request body.

For endpoint configuration, user must provide the XML payload by selecting the **raw** format.

For example:

For JSON:

```
{
  "limit": 10,
  "cursor": "abcd1234"
}
```

For XML

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:bsvc="urn:com.workday/bsvc">
  <soapenv:Header>
    <Security xmlns=
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      ...
    </Security>
  </soapenv:Header>
  <soapenv:Body>
    ...
  </soapenv:Body>
</soapenv:Envelope>
```

The following keywords will be used in the Body along with the placeholder \$:

- plan
- response
- application
- getobject
- authenticate

Handling '\$' Character In URL and Body

The Web Services connector extracts and handles the '\$' character, in an URL and body, along with the placeholders.

The following table provides the details of the placeholder:

Request	Description
URL	If the URL contains placeholder that is not present in provisioning plan, response or application, then connector removes the value from the Request URL.
Header	If the header contains a value which is not present in provisioning plan, response or application, then connector removes the key value from the Request Header.

Request	Description
<p>The JSON request with the placeholder in the Body / Payload</p>	<p>If the body contains placeholder that is not present in the provisioning plan, response or application, then connector removes the key value from the JSON payload.</p> <p>Refer to the following example:</p> <pre>{ "type": "\$plan.type\$", "id": "\$plan.nativeIdentity\$", "title": "\$plan.title\$", "externalid": "12345" }</pre> <p>If the plan does not contain the title attribute request, the connector will remove the title key and value as mentioned below:</p> <pre>{ "type": "student", "id": "STUD121", "externalid": "12345" }</pre>
<p>The XML request having placeholder in the Body / Payload</p>	<p>If the body contains placeholder that is not present in provisioning plan, response or application, then connector removes the value from the XML payload.</p> <p>Refer to the following example:</p> <pre><worker> <fname>\$plan.type\$</fname> <lname>\$plan.type\$</lname> <wid>\$plan.nativeIdentity\$</wid> <description> permanent worker</description> </worker></pre> <p>If the plan does not contain lname attribute request. The connector will remove lname value as mentioned below:</p> <pre><worker> <fname>John</fname> <lname></lname> <wid>wid121</wid></pre>

Request	Description
	<pre><description> permanent worker</description> </worker></pre>

Escaping character in Body of a SOAP Request

User can use the following format for escaping a character required to be hardcoded in the request/XML:

```
<![CDATA[*+_value_+*]]>
```

For example, `<![CDATA[CC&SD]]>` for ampersand for value CC&SD.

Response

(For XML Web Services) XPath Namespace Mapping: XML Namespace Prefix and corresponding Namespace URL identify uniquely named elements and attributes in XML request/response.

If there exists any non-standard XML Namespace in the response, configure the same in the XML Namespace mapping where the key is Namespace Prefix and value is the Namespace URL.

Absence of non-standard Xml Namespaces would result in errors while response parsing.

If a default Namespace is present, add a temporary Namespace Prefix with the default Namespace URL in the XML Namespace mappings. Further, use this temporary prefix in the XPATH elements within the scope of the default Namespace.

For example, see 'Example 1: XML response for mapping' of payload in [For XML](#).

Root Path: Common path present in the JSON/XML response.

It must be common for all the above attribute mentioned in the **Response Attribute Mapping**. Default: \$

For example,

(For JSON) \$.members.profile

(For XML) //wd:Response_Data/wd:Worker/wd:Worker_Data

Successful Response Code: Successful response code expected by the respective Web Service operation.

This field accepts HTTP status code in csv format (For example, 200, 201, 203).

If the list does not contain any value, the status code from 200 to 299 would be checked.

There could be situation where successful status code may start with 2, in this situation user can provide 2**.

Before Rule

Before Operation Rule: Rule that will be invoked before performing any operation (account aggregation, enable, disable account and so on).

After Rule

After Operation Rule: Rule that will be invoked after performing any operation (account aggregation, enable, disable account and so on)

Parent Endpoint

(Applicable only for operations of type aggregation and get object) Provide the parent endpoint name for end-point chaining. If multiple endpoints of the same operation type exists and there is no parent endpoint configuration provided, then each endpoint would behave as an independent endpoint.

For more information on operation specific configurations, see [Operation Specific Configurations](#) .

Operation Rule

Web Services uses the following operation rules:

- `WebServiceBeforeOperationRule`
- `WebServiceAfterOperationRule`

The before and after operation rules provide mechanism to implement custom code agnostic to the Web Service. If such code generates any sensitive information that would be used by the connector for operation, then such sensitive values must be added to the `LogContext` to prevent it from leaking into the logs.

Use the following code for adding the sensitive values generated by the rules to the `LogContext`:

```
{code}
import connector.common.logging.LogContext;
...
LogContext.addSensitiveValue("some-secret-value");
{code}
```

For more information on Web Services Before/After Operation Rule, see [Web Services Before/After Operation Rule](#).

Saving Parameters in Web Services Connector

Web Services Connector supports storing the values in application object permanently. Saving of the parameters can be configured using the `connectorStateMap` in BEFORE and AFTER operation rules of Web Service Connector. Following are the examples of BEFORE and AFTER operation rules.

BEFORE Operation Rules

```
Map updatedInfoMap = new HashMap();
requestEndPoint.setFullUrl(requestEndPoint.getFullUrl().replaceAll("&&", "&"));
Map connectorStateMap = new HashMap();
connectorStateMap.put("accesstoken", "Bearer accessTokenGeneratedInBeforeRuleScript");
updatedInfoMap.put("updatedEndPoint", requestEndPoint);
updatedInfoMap.put("connectorStateMap", connectorStateMap);
return updatedInfoMap;
```

AFTER Operation Rule

```
import java.util.*;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Iterator;
Map updatedMapInfo = new HashMap();
```

```

if (parsedResponseObject != null){
    System.out.println("Parsed response is not null");
    for (Map iterateMap : parsedResponseObject) {
        if (iterateMap != null ) {
            Set keySet = iterateMap.keySet();
            for (String s : keySet) {
                System.out.println(s);
                if (s.equals("given_name")) {
                    String forStr = (String) iterateMap.get("given_name");
                    forStr = "TEST"+ forStr;
                    System.out.println("forStr: " + forStr );
                    iterateMap.put("given_name", forStr);
                }
            }
        }
    }
    updatedMapInfo.put("data", parsedResponseObject);
}
Map connectorStateMap = new HashMap();
connectorStateMap.put("refresh_token", "refreshTokenGeneratedInAfterRuleScript");
updatedInfoMap.put("connectorStateMap", connectorStateMap);
return updatedMapInfo;

```

Keywords

Web Service supports the following keywords for various configuration attributes such as context URL, Headers, Body (JSON and Form-data) for a single or multiple endpoints:

plan

Used for configuring the provisioning operations such as, create account, update account among others.

For example:

(Context URL): /api/core/v3/people/\$plan.nativeIdentity\$

(JSON Body example for plan):

```

{"new_members": [
  {
    "email": "$plan.email$",
    "first_name": "$plan.first_name$",
    "surname": "$plan.surname$",
    "send_welcome_email": $plan.send_welcome_email$,
    "role": {
      ".tag": "member_only"
    }
  }
]
}

```

response

Used for multiple endpoints, where the response from the first endpoint is provided as an input for the second endpoint.

For example, there are two endpoints for account aggregation.

- The first endpoint returns a response as a list of **member_ids** that is an input for the second endpoint as mentioned in the next point.
- Second endpoint's JSON body is: {"members_info":[{"member_id":"\$response.member_id\$"}]}

application

Used to get other configuration attributes from the current application.

For example

```
"$application.accesstoken$",
```

where the accesstoken is an application configuration attribute.

getobject

Used while performing **Aggregate Account** (get a single account details).

For example:

- **(JSON body):**

```
{
  "members": [
    {
      ".tag": "member_id",
      "member_id": "$getobject.nativeIdentity$"
    }
  ]
}
```

- **(Context URL for get object):**

```
/api/v4/admin/$getobject.nativeIdentity$
```

nativeidentity

Signifies the **AccountID** (identity attribute) in the plan or during **getobject** operation.

For example, nativeidentity would be used along with the keyword as follows:

- **getobject:** \$getobject.nativeIdentity\$
- **plan:** \$plan.nativeIdentity\$

planNativeidentity

Signifies the **Plans** nativeidentity in the Provisioning plan.

For example, planNativeidentity would be used along with the keyword as follows:

- **plan:** \$plan.planNativeIdentity\$

Web Services can be configured with native identity from the provisioning plan using placeholder \$plan.planNativeIdentity\$.

authenticate

To provide username and password in endpoint configuration user can use the following placeholders:

```
$authenticate.username$
```

```
$authenticate.password$
```

For more information on configuration for pass-through authentication, see [Pass Through Authentication](#).

SailPoint recommends to use placeholder in the body and url rather than adding sensitive information directly. For example, **`https://TESTMACHINE:9096/users/user/$application.accesstoken$`**

In the above table for examples of attributes that are mapped to a raw JSON response, it may contain formatted values as follows (similar to ".tag": "member_id"):

```
['.tag']  
['@etag']  
['@@test']  
['complex.name']  
['role name']  
['role_name']
```

Supported Operations

This section describes the various supported operations of Web Services. For certain operations, the Body must be updated accordingly with operation specific configuration parameters. For more information, see [Operation Specific Configuration Parameters](#).

Test Connection Aggregation

The Web Services Connector supports aggregation using the following mechanisms:

- Parent-Child Configuration
- Or
- Multiple Independent Endpoints

For more information on multiple independent endpoints, see [Multiple Independent Endpoints](#).







Configuration for Multiple Endpoints

Perform the following to obtain the properties of account/group/Get Object from multiple endpoints:

1. The basic attribute is obtained from the first endpoint and is then used for fetching the data from rest of the endpoints.

For example, during aggregation of Jive some attributes are obtained from first endpoint ([Mapped Schema Attribute](#)) using the following URL: <https://myDomain.jive.com/api/core/v3/people>

2. **Account Aggregation - 1**
 - a. As displayed in the following screenshot, under **Operations**, enter the name of the operation as Account Aggregation -1 in **Operation Name**, select Account Aggregation from the **Operation Type** drop-down list. Click **Add New Operation**.

Connector Operations							Add Operation
Order	Operation	Name	Status	Before Rule	After Rule	Actions	
1	Test Connection	Test Connection	Configured	<input type="checkbox"/>	<input type="checkbox"/>	 	
2	Account Aggregation	Account Aggregation-1	Configured	<input type="checkbox"/>	<input type="checkbox"/>	 	
3	Account Aggregation	Account Aggregation-2	Configured	<input type="checkbox"/>	<input type="checkbox"/>	 	

Test Connection

- Enter the **Context URL** and select the method from the **HTTP Method drop-down** list.
- Mapped schema attributes.

Back
Connection Settings

Context URL: /api/core/v3/people
Method: GET

Header
Body
Response
Before Rule
After Rule
Paging
Parent Endpoint

Response Attribute Mapping
Add Row

Schema Attribute	Attribute Path
country	addresses[0].value.locality
profileValue	jive.profile[0].value
displayName	displayName
formatted	name.formatted
id	id
value	email[0].value
phoneNo	phoneNumbers[0].value
status	jive.enabled

XPath Namespace Mapping (For XML web services)
Add Row

Root Path: \$.list
Successful Response Code: 2**

3. Account Aggregation - 2

- To fetch additional attribute from another endpoint use the `id` attribute from the previous response. Add these attributes in Schema Attribute of Response Attribute Mapping and response as follows:

Response: The following context URL contains `id` which fetches all the groups connected to that account:

[https://myDomain.jive.com/api/core/v3/people/\\$response.id\\$/securityGroups](https://myDomain.jive.com/api/core/v3/people/$response.id$/securityGroups)

- Mapped Schema Attribute

Context URL [?]

Method [?]

Response Attribute Mapping Add Row

Schema Attribute	Attribute Path
group_id	id
group_name	name

XPath Namespace Mapping (For XML web services) Add Row

Root Path [?]

Successful Response Code [?]

- c. In the above example Account Aggregation -1 is the Parent Endpoint and Account Aggregation -2 is its Child Endpoint.
Hence enter the parent endpoint operation name- Account Aggregation -1 in its child endpoints - **Parent Endpoint** tab as shown below:

Context URL [?]

Method [?]

Parent Endpoint Configuration for Chained Endpoints

Parent Endpoint Name

Create Account

The Web Services, at first, creates an account on the managed system and then adds entitlements one by one. If **GetObject** is configured, the connector will invoke the **GetObject** endpoint with the respective identity attribute. The identity attribute can be fetched through provisioning plan or the response returned by the managed system.

If **GetObject** operation is present, the Web Services would invoke **GetObject** operation using identity attribute or it would directly update resource object in provisioning plan (provisioning Result).

If the managed system supports a single/ separate endpoint for creating an account and adding an entitlement, use the **createAccountWithEntReq** attribute.

Following is an example for updating the Body for create account in Dropbox. For fetching attribute through Provisioning Plan, the body must be updated in the following manner. This fetches the attribute detail through Provisioning Form and updates the endpoint.

- (For JSON) In the following Body,
 - **\$plan** represents the Provisioning Plan that is passed to provision method
 - **\$plan.member_surname**: the connector checks for **member__surname** in the attribute request and updates in the body after it is found

Body

☐ form-data
 ☒ raw

```

{
  "member_email" : "$plan.member_email$",
  "member_given_name" : "$plan.member_given_name$",
  "member_surname" : "$plan.member_surname$",
  "send_welcome_email" : "$plan.send_welcome_email$",
  "member_external_id" : "$plan.member_external_id$"
}

```

- (For XML) To create account for XML payload:

Body

☐ form-data
 ☒ raw

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:partner.soap.sforce.com"
xmlns:urn1="urn:object.partner.soap.sforce.com">
  <soapenv:Header>
    <urn:SessionHeader>
      <urn:sessionId>$application.accesstoken$</urn:sessionId>
    </urn:SessionHeader>
  </soapenv:Header>
  <soapenv:Body>
    <urn:create>
      <!--Zero or more repetitions-->
      <urn:sObjects>
        <urn1:type>User</urn1:type>
        <!--Zero or more repetitions-->
        <urn1:Username>$plan.Username$</urn1:Username>
        <urn1:LastName>$plan.LastName$</urn1:LastName>
        <urn1:FirstName>$plan.FirstName$</urn1:FirstName>
        <urn1:Email>$plan.Email$</urn1:Email>
        <urn1:Alias>$plan.Alias$</urn1:Alias>
        <urn1:CommunityNickname>$plan.CommunityNickname$</urn1:CommunityNickname>
        <urn1:IsActive>true</urn1:IsActive>
        <urn1:TimeZoneSidKey>America/Los_Angeles</urn1:TimeZoneSidKey>
        <urn1:LocaleSidKey>en_US</urn1:LocaleSidKey>
        <urn1:LanguageLocaleKey>en_US</urn1:LanguageLocaleKey>
        <urn1:ProfileId>00ei00000000ye0AAQ</urn1:ProfileId>
        <urn1:EmailEncodingKey>UTF-8</urn1:EmailEncodingKey>
        <!--You may enter ANY elements at this point-->
      </urn:sObjects>
    </urn:create>
  </soapenv:Body>
</soapenv:Envelope>

```

To get object for XML payload

Body

☐ form-data
 ☒ raw

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:partner.soap.sforce.com">
  <soapenv:Header>
    <urn:SessionHeader>
      <urn:sessionId>$application.accesstoken$</urn:sessionId>
    </urn:SessionHeader>
  </soapenv:Header>
  <soapenv:Body>
    <urn:query>
      <urn:queryString>Select Id , Alias , City , CommunityNickname , CompanyName , CallCenterId , Country , Department , Email ,
      Division , EmployeeNumber , Extension , Street , Fax , IsActive , Username , FirstName , LastName , EmailEncodingKey , Name ,
      UserPermissionsMarketingUser , UserPermissionsMobileUser , UserPermissionsOfflineUser , UserPermissionsSFContentUser , Phone ,
      ProfileId , Profile.Name , ReceivesAdminInfoEmails , UserRoleId , UserRole.Name , UserType , State , Title , ReceivesInfoEmails , Profile.Id ,
      UserRole.Id from user Where Id = '$getObject.nativelidentity$'
    </urn:queryString>
  </urn:query>
</soapenv:Body>
</soapenv:Envelope>
```

Create Account Scenarios

Entitlements	GetObject	Description of the Scenario
No	No	<ul style="list-style-type: none"> The Create Account operation is executed. If the response of create account contains the required attributes including the identity attribute, the account object would be created and account link would be displayed under the identity from where the account creation was triggered.
No	Yes	<ul style="list-style-type: none"> The Create Account operation is executed. If the response of create account contains the identity attribute or the request (plan) contains identity attribute, the GetObject would be triggered using the respective identity attribute.
Yes	No	<ul style="list-style-type: none"> When createAccountWithEntReq is set to false, the create account operation would be executed without entitlement attributes. All the entitlement attributes would be executed using the Add Entitlement operations. For example, if there are four entitlements, Add Entitlement operation would be executed four times.
Yes	Yes	<ul style="list-style-type: none"> When createAccountWithEntReq is set to false, the create account operation would be executed without entitlement attributes. All the entitlement attributes would be executed using the Add Entitlement operations. For example, if there are four entitlements, Add Entitlement operation would be executed four times. The GetObject operation would be executed updating the complete account object even the entitlement attributes also, if the GetObject response returns entitlement attributes.



Entitlements	GetObject	Description of the Scenario
Yes	No	<ul style="list-style-type: none"> When createAccountWithEntReq is set to true, the create account operation would be executed with entitlement attributes. If the response of create account operation contains the required attributes including the identity attribute, the account object would be created and account link would be displayed under the identity from where the account creation was triggered.
Yes	Yes	<ul style="list-style-type: none"> When createAccountWithEntReq is set to true, the create account operation would be executed with entitlement attributes. If the response of the create account operation contains the identity attribute or the request (plan) contains identity attribute, the GetObject would be triggered using the respective identity attribute.

Enable/ Disable

Set the get object endpoint for enable/ disable operation as in the POST method the complete object would be required to update and not single attribute. Hence first endpoint getObject would fetch the whole account and later the endpoint would update the payload with all the required attributes using the response of the first endpoint.

Perform the following steps to get object for Enable operation with PUT method

1. Configure the first endpoint to get object for Enable.

^ v	Enable Account ▾	Enable ONE - getObject	Configured	<input type="checkbox"/>	<input type="checkbox"/>		
^ v	Enable Account ▾	Enable TWO - enable	Configured	<input type="checkbox"/>	<input type="checkbox"/>		

2. Configuration for the first endpoint:

Context URL ?

/api/core/v3/people/\$getObject.nativeIdentity\$

Method ?

GET

Header

Body

Response

Before Rule

After Rule

Response Attribute Mapping

Add Row

Schema Attribute ?	Attribute Path ?	
country	addresses[0].country	X
Email	emails[0].value	X
displayName	displayName	X
street	addresses[0].value.streetAddress	X
givenName	name.givenName	X
name	displayName	X
id	id	X
region	addresses[0].value.region	X
type	type	X
status	jive.enabled	X

This endpoint retrieves getObject for account for which Provisioning Operation is performed.

- Configuration for second endpoint for Enable endpoint as shown in the following figure:

Context URL ?

/api/core/v3/people/\$plan.nativeIdentity\$

Method ?

PUT

Header

Body

Response

Before Rule

After Rule

form-data

raw

```

{
  "emails": [
    {
      "value": "$response.Email$",
      "jive_label": "Email"
    }
  ],
  "jive": {
    "enabled": "true",
    "federated": "false",
    "visible": "true",
    "username": "$response.Email$"
  },
  "name": {
    "formatted": "$response.givenName$",
    "familyName": "$response.familyName$",
    "givenName": "$response.givenName$"
  }
}

```

It may be required to update few attribute for performing enable/ disable operation

Similar steps are to be performed for Disable operation.

Add/ Remove Entitlement

Following is an example of the Body entry for Add Entitlement:

Context URL ?

/1/team/groups/members/add

Method ?

POST

Header

Body

Response

Before Rule

After Rule

Body

form-data ☒ raw

```
{
  "group_id": "$plan.groups$",
  "members": [
    { "team_member_id": "$plan.nativeIdentity$",
      "access_type": "member" } ]
}
```

On similar basis as above example the Body entry must be updated for Remove Entitlement.

Update Account

Following is an example of the Body entry for Update Account:

Context URL ?

/1/team/members/set_profile

Method ?

POST

Header

Body

Response

Before Rule

After Rule

Body

form-data ☒ raw

```
{
  "member_id": "$plan.nativeIdentity$",
  "new_given_name": "$plan.given_name$",
  "new_email": "$plan.email$"
}
```

Delete Account

Following is an example of the Body entry for Delete Account:

Context URL ?

/1/team/members/remove

Method ?

POST

Header

Body

Response

Before Rule

After Rule

Body

form-data ☒ raw

```
{
  "member_id": "$getobject.nativeIdentity$"
}
```

Change Password

Following is an example of the Body entry for Change Password:

The screenshot shows the 'Configuration' tab in the SailPoint interface. Under 'Settings', the 'Connection Settings' section is active. It displays a 'Context URL' of '/api/now/v1/import/x_sapo_iiq_connect_sysuser' and a 'Method' of 'POST'. On the left, a sidebar lists 'Header', 'Body', 'Response', 'Before Rule', and 'After Rule'. The 'Body' tab is selected, showing a 'form-data@raw' input field with the JSON payload: `{"user_sys_id":"$plan.nativeIdentity$","password":"$plan.password$","password_needs_reset":"false"}`. 'Cancel' and 'Save' buttons are at the bottom right.

Configuration for Response

When configuring the Web Services application, map the schema attribute as follows for JSON and XML.

For JSON

- **Single endpoint:** Refer the following example:
Example for mapping the schema attributes with JSON

```

{
  "list": [
    {
      "id": "2124",
      "resources": {
        "securityGroups": {
          "ref": "https://mydomain.jive.com/api/core/v3/people/2124/securityGroups"
        },
        "displayName": "Bill Jackson",
        "emails": [
          {
            "value": "bill.jackson@mydomain.com",
          },
          {
            "value": "admin@mydomain.com",
          }
        ],
        "jive": {
          "enabled": true,
          "level": {
            "name": "Level 0",
          },
          "username": "bill.jackson",
        },
      },
    },
  ],
}

```

In the above JSON response, all the attributes can be mapped as follows considering Root Path as \$.list:

Id = id

displayName=displayName

username=jive.username

```
enabled =jive.enabled
```

```
emails=emails[*].value
```

- **Mapped schema attributes**

Response Attribute Mapping

Schema Attribute ?	Attribute Path ?
id	id
enabled	jive.enabled
username	jive.username
emails	emails[*].value
displayName	displayName

XPath Namespace Mapping (For XML web services)

Root Path ?
\$.list

Successful Response Code ?
2**|

- **Multiple endpoint:** Refer to the following response example for first and second endpoint:

Type	Response Example	Response Mapping
First end-point	<pre>{ "meta": { "query_time": 0.141557942, "powered_by": "csam", "trace_id": "8fe96ed0-4da3-42ab-b97a-4651e39f7d67" }, "resources": ["01dcc3dd-5b72-4fb2-a43b-424e092bd6af", "032b704e-1467-48b3-ad70-29a04f1a20ab", "042ab092-918f-4ed3-97eb-196fbe917e77",], "errors": [] }</pre>	Response Attribute Mapping: uid = \$ Root Path: \$.resources.*
Second endpoint	Context URL: /users/entities/users/v1?ids=\$response.uid\$ Example:	Response Attribute Mapping: firstName = \$.resources

Type	Response Example	Response Mapping
	<pre> { "meta": { "query_time": 0.042584188, "powered_by": "csam", "trace_id": "0973beb6-8afe-452c-bac4-18d17cc3e9ef" }, "resources": [{ "uid": "john.doe@santander.us", "firstName": "John", "lastName": "Doe", "uuid": "01dcc3dd-5b72-4fb2-a43b-424e092bd6af", "customer": "81971f9d094d494bae5a5fd97052509f" }], "errors": [] } </pre>	<pre> [0].firstName lastName = \$.resources [0].lastName </pre> <p>Root Path: Root path not required</p>

- **Parsing JSON Array:** Following is an example for parsing JSON array:

```
[1,10, 21, 31,46, 89,90,67,88,98,101]
```

The response mapping for above example is:

Response Attribute Mapping (id is attribute in the schema): id = \$

- Root Path: \$.*

For XML

- **Example 1:** XML response for mapping:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="urn:partner.soap.sforce.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:sf="urn:object.partner.soap.sforce.com">
  <soapenv:Header>
    .....
  </soapenv:Header>
  <soapenv:Body>
    <queryResponse>
      <result xsi:type="QueryResult">
        <done>true</done>
        <queryLocator xsi:nil="true"/>
        <records xsi:type="sf:sObject">
          <sf:type>User</sf:type>
          <sf:Id>123456</sf:Id>
          <sf:Id>123456</sf:Id>
          <sf:Alias>Alias</sf:Alias>
          <sf:City xsi:nil="true"/>
          <sf:CommunityNickname>CName1</sf:CommunityNickname>
          <sf:Email>test@test.com</sf:Email>
          <sf:IsActive>false</sf:IsActive>
          <sf:Username>test@test.com</sf:Username>
          <sf:FirstName>Test</sf:FirstName>
          <sf:LastName>Test</sf:lastName>
        </records>
        <size>1</size>
      </result>
    </queryResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

See the following figure which mentions the XPath Namespace mapping for XML Web Services:

Schema Attribute	Attribute Path
ProfileId	sf:ProfileId
PublicGroups	sf:PublicGroups/sf:Name
firstName	sf:FirstName
PermissionSet	sf:PermissionSet/sf:Name
UserRoleId	sf:UserRoleId
Username	sf:Username
Alias	sf:Alias
IsActive	sf:IsActive
ProfileName	sf:Profile/sf:Name
LastName	sf:LastName
id	sf:Id[1]
email	sf:Email

Namespace Prefix	Namespace URI
sf	urn:subject.partner.soap.sforce.com
soapenv	http://schemas.xmlsoap.org/soap/envelope/
ns	urn:partner.soap.sforce.com
xsi	http://www.w3.org/2001/XMLSchema-instance

Root Path: //ns:result/ns:records

Successful Response Code: 2**

- **Example 2:** XML response for mapping:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><userList id="5457373">
<name>DillardsEveryone</name>
<memberList>
<listMember>drazzetti@ravemobilesafety.com</listMember>
<listMember>gage5.test5@test5.dillards.com</listMember>
</memberList>
</userList>
```

See the following figure which mentions the Root Path configuration for listMember:

Schema Attribute	Attribute Path
name	text()

Root Path: //userList/memberList/ListMember

Successful Response Code: 200

Additional Configuration Parameters

Add the following attributes in the application debug page:

throwBeforeAfterRuleException

During aggregation if an exception is displayed from **WebServiceBeforeOperationRule** or **WebServiceAfterOperationRule**, then aggregation continues and completes successfully.

Set the value of the following flag to **true** to terminate the aggregation by displaying an error message:

```
throwBeforeAfterRuleException
```

This flag can be set only for Account and Group aggregation (multiple group aggregation if any).

The default value of the `throwBeforeAfterRuleException` flag is set to false.

This flag is not applicable for child endpoints.

throwProvBeforeRuleException

During Provisioning, GetObject and Test Connection, if an exception is thrown from **WebServiceBeforeOperationRule**, then Provisioning would fail. Hence to dispose of the exception in the log file and proceed with provisioning, set the value of **throwProvBeforeRuleException** to false as follows:

```
<entry key="throwProvBeforeRuleException">
  <value>
    <Boolean>true</Boolean>
  </value>
</entry>
```

The default value of 'throwProvBeforeRuleException' flag is set to true for new Web Services application and false for existing application (before upgrading to IdentityIQ version 8.0 Patch 5 and above). The **throwProvBeforeRuleException** flag can be set for all operations except Account and Group aggregation.

throwProvAfterRuleException

During Provisioning, GetObject and Test Connection if an exception is thrown from **WebServiceAfterOperationRule**, then Provisioning would fail. Hence to dispose of the exception in the log file and proceed with provisioning, set the value of **throwProvAfterRuleException** to false as follows:

```
<entry key="throwProvAfterRuleException">
  <value>
    <Boolean>true</Boolean>
  </value>
</entry>
```

The default value of 'throwProvAfterRuleException' flag is set to true for new Web Services application and false for existing application (before upgrading to IdentityIQ version 8.0 Patch 5 and above). The **throwProvAfterRuleException** flag can be set for all operations except Account and Group aggregation.

createAccountWithEntReq

Default value: false

To enable the functionality of sending entitlements with create account in a single request to the managed system, set the value of **createAccountWithEntReq** parameter to true as follows:

```
<entry key="createAccountWithEntReq">
  <value>
    <Boolean>true</Boolean>
  </value>
</entry>
```

enableHasMore

If **enableHasMore** is set to true as follows then the termination of aggregation would depend on the value of **hasMore** attribute:

```
<entry key="enableHasMore" value="true"/>
```

The **hasMore** attribute is the boolean attribute which is to be set in the **transientValues** map in the before/after operation rule. Unless the value of **hasMore** attribute is false aggregation would not be terminated.

If **enableHasMore** is set to false as follows, then the aggregation would be terminated if the number of accounts returned is zero:

```
<entry key="enableHasMore" value="false"/>
```

lockStatus

To utilize the Lock/Unlock feature, add the **lockStatus** attribute in the application debug page as provided in the following example:

```
<entry key="lockStatus" value="status=inactive"/>
```

In the above example, **status** is an account schema attribute that indicates if the account is locked or not. The value (inactive) helps the Web Services Connector to distinguish between the lock or unlock account.

The Web Services Connector does not support provisioning of lock account.

possibleHttpErrors

When an API endpoint does not send expected error codes to flag failure conditions, the connector can be configured as follows (example) with all possible HTTP error codes/ messages, which the API endpoint would return resulting into failure of operations:

```
<entry key="possibleHttpErrors">
  <value>
    <Map>
      <entry key="errorCodes">
        <value>
          <List>
            <Integer>401</Integer>
          </List>
        </value>
      </entry>
      <entry key="errorMessages">
        <value>
          <List>
            <String>Unauthorized</String>
          </List>
        </value>
      </entry>
    </Map>
```

```
</value>
</entry>
```

In few instances the Web Services Connector returns `httpstatuscode` as 200 but the response payload may contain error. In this case, ideally the connector must fail the request or OAuth token generation must try to regenerate the token

These possible `HttpError` codes/ messages can also be configured to specify invalid/ expiry token error. In this case connector would regenerate and save the token for OAUTH2 authentication and retries the operation with the newly generated access token.

This flag is not applicable for child endpoints.

isGetObjectRequiredForPTA

For using the Web Service as a Pass-through Authentication Connector, set the value of **`isGetObjectRequiredForPTA`** to true as follows:

```
<entry key="isGetObjectRequiredForPTA">
  <value>
    <Boolean>true</Boolean>
  </value>
</entry>
```

For new Web Services application created, default value for **`isGetObjectRequiredForPTA`** would be set to true.

When set to true, it would execute Get Object operation to verify if the entered `userName` (Considered as Identity attribute) is present on the managed system or not.

When set to false then it would skip Get Object operation and Pass-through Authentication operation must have response mapping with account object schema attributes.

For more information, see [Pass Through Authentication](#).

objectNotFoundErrMsg

Based on the error message list, Connector would decide to display the **`objectNotFoundErrMsg`** error.

For example, user can create the following entry for **`objectNotFoundErrMsg`** with custom error message to identify exceptions (these can be multiple):

```
<entry key="objectNotFoundErrMsg">
  <value>
    <List>
      <String>404: Not Found</String>
      <String>404</String>
    </List>
  </value>
</entry>
```

authenticationFailedErrMsg

Based on the error message list, Connector would decide to display the **`authenticationFailedErrMsg`** error.

For example, user can create the following entry for **authenticationFailedErrorMsg** with custom error message to identify exceptions (these can be multiple):

```
<entry key="authenticationFailedErrorMsg">
  <value>
    <List>
      <String>Authentication Failed</String>
    </List>
  </value>
</entry>
```

expiredPasswordErrorMsg

Based on the error message list, Connector would decide to throw **expiredPasswordErrorMsg** error.

For example, user can create the following entry for **expiredPasswordErrorMsg** with custom error message to identify exceptions (these can be multiple):

```
<entry key="expiredPasswordErrorMsg">
  <value>
    <List>
      <String>Password Expired</String>
    </List>
  </value>
</entry>
```

If response contains string matched with **expiredPasswordErrorMsg**, then it would redirect user from login page to Change Password page.

updateAttrWithChangePassword

To enable single request for change password and update operations, add **updateAttrWithChangePassword** attribute as follows and set the value to true:

```
<entry key="updateAttrWithChangePassword" value="true"/>
```

If the value of **updateAttrWithChangePassword** attribute is true in the application and plan contains no attribute request name as password (change password), then specific update operation type endpoint would be invoked.

Setting the value of this attribute to true would not include requests for add/ remove entitlement. These requests would be executed independently.

disableCookies

By default, the Web Services Connector supports cookies. Cookies can be disabled by adding the **disableCookies** attribute as follows and set the value to true:

```
<entry key="disableCookies" value="true"/>
```

Not applicable for the authentication API call operation.

addRemoveEntInSingleReq

Add or remove multiple entitlements such as role or profile.

skipGetObjectInCreate

Webservices Connector now supports the **skipGetObjectInCreate** attribute (Boolean) to skip getObject call if it is present during create provisioning operation.

Additional Configuration Parameters for OAuth2

Following attributes are applicable only if **Authentication Method** is selected as OAuth2:

oauth_headers

To have customized headers as a part of the access token generation request, add the **oauth_headers** parameter as follows:

```
<entry key="oauth_headers">
  <value>
    <Map>
      <entry key="Content-Type" value="application/x-www-form-urlencoded" />
    </Map>
  </value>
</entry>
```

Web Services now uses access token configured in the application as authorization header for each endpoint, users would no longer require to specify the authorization header for each endpoint. If authorization is provided at endpoint level then it would precede over the access token.

SailPoint recommends to provide authorization header suffix in the access token provided. For example, `Bearer <Access Token>`. If no prefix is provided, then Web Services would by default provide `Bearer` as Authorization header prefix.

To send additional headers for token generation, add the **oauth_headers** parameter as follows:

```
<entry key="oauth_headers">
  <value>
    <Map>
      <entry key="customHeaderKey" value="customHeaderValue"/>
    </Map>
  </value>
</entry>
```

oauth_headers_to_exclude

Web Services supports exclusion of headers in the OAuth2 request. The header keys for headers which are intended to be excluded from the OAuth2 request, can be added as comma separated values as follows:

```
<entry key="oauth_headers_to_exclude" value="Authorization,CUSTOM_HEADER"/>
```

oauth_request_parameters

For some managed systems, custom request parameters may be required to be part of the access token generation request. To send additional parameters for token generation, add the following entry:

```
<entry key="oauth_request_parameters">
  <value>
    <Map>
      <entry key="customParamKey" value="customParamValue"/>
    </Map>
  </value>
</entry>
```

```
</value>
</entry>
```

oauth_body_attrs_to_exclude

To delete any of the standard request parameters which are not supported by managed systems access token generation requests, add the following entry:

```
<entry key="oauth_body_attrs_to_exclude">
  <value>
    <Map>
      <entry key="oauth_body_attrs_to_exclude"
value="customParamKey1,customParamKey2"/>
    </Map>
  </value>
</entry>
```

saml_headers

(Optional) Used for SAML assertion generation, if additional header information is required then add the following entry:

```
<entry key="saml_headers">
  <value>
    <Map>
      <entry key="customHeaderKey" value="customHeaderValue"/>
    </Map>
  </value>
</entry>
```

saml_headers_to_exclude

(Optional) Used for SAML assertion generation, the key provided here would be excluded from the header while executing SAML Assertion generation request. The keys that need to be excluded can be added as comma separated values as follows:

```
<entry key="saml_headers_to_exclude" value="Authorization,CUSTOM_HEADER"/>
```

oAuthJwtHeader

Contains the alg (algorithm that is used for signing the JWT assertion) as follows:

```
<entry key="oAuthJwtHeader">
  <value>
    <Map>
      <entry key="alg" value="RS256"/>
    </Map>
  </value>
</entry>
```

If required additional header attributes can be provided in this map.

oAuthJwtPayload

Contains the aud (Audience), Expiry of the JWT assertion (exp), iss (Issuer), sub (Subject) as follows:

```

<entry key="oAuthJwtPayload">
  <value>
    <Map>
      <entry key="aud" value=""/>
      <entry key="exp" value="15f"/>
      <entry key="iss" value=""/>
      <entry key="sub" value=""/>
    </Map>
  </value>
</entry>

```

If required additional payload attributes can be provided in this map. For additional attributes like `jti`, `iat`, `nbf` if only `key` (not `value`) is available in the map then it would consider the default values for the same.

Application Layer Proxy Support

The Webservice Connector supports an Application layer Proxy which when configured for a specific application to ensure that the traffic is routed through the proxy server.

Add following lines in Application XML based on the requirement.

HTTPS Proxy:

```

<entry key="https.proxyHost" value="IP of Proxy Server"/>
<entry key="https.proxyPort" value="Port on which Proxy is configured"/>

```

HTTP Proxy:

```

<entry key="http.proxyHost" value="IP of proxy Server"/>
<entry key="http.proxyPort" value="Port on which proxy Server is configured."/>

```

For enabling Authentication with HTTPS proxy:

```

<entry key="https.proxyUser" value="Username"/>
<entry key="https.proxyPassword" value="password"/>

```

For enabling Authentication with HTTP proxy:

```

<entry key="http.proxyUser" value="Username"/>
<entry key="http.proxyPassword" value="password"/>

```

Refer to the following API for updating the source <https://developer.sailpoint.com/apis/v3/#operation/replaceSource>

Configuring Web Services for Integration

This section describes the various configurations to be performed to support the following features:

- [Pass Through Authentication](#)
- [Multiple Independent Endpoints](#)
- [Pagination](#)
- [Partitioning](#)
- [Configuration for Response](#)
- [Enabling Single API Calls](#)
- [Configuring Multiple Entitlement Requests](#)

Pass Through Authentication

To Configure Pass Through Authentication on existing Web Services, perform the following:

1. Add **AUTHENTICATE** in the featureString of the application debug page.
For more information on authenticate, see Keywords.
2. Add Pass Through Authentication operation in Web Services application.
This operation would be used to perform verification of user credentials provided from Login page or IdentityIQ Console.
3. Add the **isGetObjectRequiredForPTA** entry key with value as **true** in the application debug page.
4. (Optional) If user wants to configure error messages for Pass Through Authentication, it can be done using the following entry keys:
 - **objectNotFoundErrMsg**
 - **authenticationFailedErrMsg**
 - **expiredPasswordErrMsg**

For more information, see [Additional Configuration Parameters](#).

For example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:partner.soap.sforce.com">
  <soapenv:Header>
    <urn:LoginScopeHeader>
      <urn:organizationId></urn:organizationId>
      <!--Optional:-->
      <urn:portalId></urn:portalId>
    </urn:LoginScopeHeader>
  </soapenv:Header>
  <soapenv:Body>
    <urn:login>
```



```
<urn:username>$authenticate.username$</urn:username>
<urn:password>$authenticate.password$</urn:password>
</urn:login>
</soapenv:Body>
</soapenv:Envelope>
```

Multiple Independent Endpoints

With IdentityIQ version 8.0 Patch 5 and onwards, Web Services allows aggregation from multiple independent endpoints. The Web Service depends upon the **Operation Type** and the **ParentEndpoint** configuration to determine whether the endpoints of type Aggregation or Get Object must be executed independently.

- For Aggregation and Get Object endpoints, when the attributes must be fetched from multiple endpoints in a chained manner, the administrator must configure the parent endpoint name in every child endpoint. There has to be a dependency between the first and subsequent endpoints. Parent endpoint will have to be explicitly configured using **Parent Endpoint** configuration tab wherein the parent endpoint's name would have to be configured for establishing the chaining of sequence of child endpoints.
- If multiple child endpoints are configured for any given endpoint, then the sequence of execution of child endpoints would be governed by the sequence in which they have been configured in the application.

For example, if account aggregation requires attributes from four different endpoints (say Ep1, Ep2, Ep3 and Ep4). Then,

- Ep1 fetches userId and primary email
- Ep2 fetches personal information using userId
- Ep3 fetches skill id information using primary email and
- Ep4 fetches skill name using skill Id from Ep3

Where **EP** = Endpoint and **RO** = Resource Object/Account object

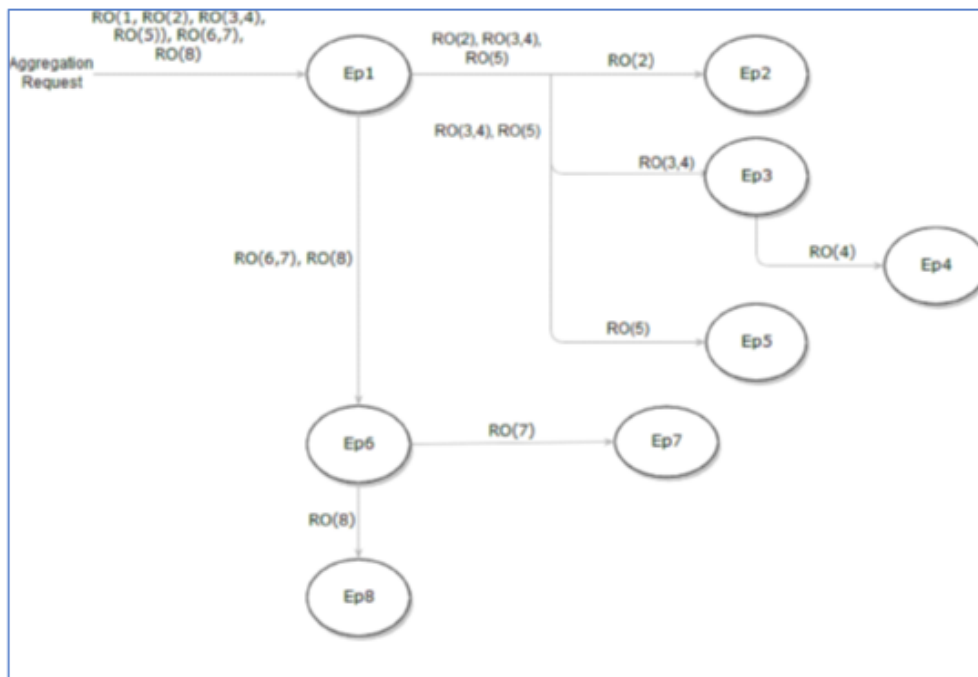
Configuring Multiple Independent Endpoints for Aggregation

- For a target system where accounts are required to be fetched from multiple endpoints and there is no relationship between such endpoints, Web Services can be configured to fetch information from independent endpoints.
- Each endpoint can be configured independently using the same **Operation Type**. No parent endpoint configuration is required.
- Every independent endpoint can have a paging of its own, if any.
- Every endpoint will collect resource objects from the configured Web Service. Upon completion of collecting all information, the next independent endpoint will be triggered for collecting resource objects.

For example, if there are two endpoints Ep1 and Ep2, then Ep1 fetches accounts from a certain geographic loc-

ation and Ep2 fetches from another.

Endpoint Name	Child Endpoint(s)	Returns	
Ep1	Ep2, Ep3, Ep5	RO(1, RO(2), RO(3,4), RO(5)), RO(6,7), RO(8)	Independent
Ep2	-	RO(2)	Child of Ep1
Ep3	Ep4	RO(3,4)	Child of Ep1
Ep4		RO(4)	Child of Ep3
Ep5		RO(5)	Child of Ep1
Ep6	Ep7	RO(6,7), RO(8)	Independent
Ep7		RO(7)	Child of Ep6
Ep8		RO(8)	Independent



The **Parent Endpoint** configuration tab would be displayed for application created after 8.0 Patch 5 and onwards. However, any changes to the parent configuration will have no effect on such applications and the operations would behave as per the previous endpoint chaining rules that is, the first endpoint of the specific operation type will be the parent and subsequent endpoints of the same operation type will be its child.

Pagination

Web Services Connector supports generic paging for Account and Group Aggregations.

Following are the methods for embedding paging in Web Service Connector:

- Using BEFORE and AFTER operation rules

Or

- Using Paging tab

BEFORE and AFTER operation rules

To embed pagination in Web Service Connector, manual processing is required in BEFORE and AFTER operation rules of Web Service Connector.

1. The Web Service Connector relies on a temporary information stored in the application object in form of a map which has the name as **transientValues**.
2. The administrator must write the Before Rule and AFTER Rule for account/group aggregation as follows:
 - **Web Service Before Rule:** The Before Rule alters the URL/request parameters if the value of the **hasMore** parameter is set to **TRUE** and the request to fetch further accounts is triggered. If **hasMore** parameter is not set or is set to **FALSE** the pagination request would be terminated.

For example, see sample Before Rule for account aggregation request in Web Services Connector for Dropbox using V2 in **examplerules.xml** file by name **Example WSBeforeRI DropboxPaging** as follows:

```
import sailpoint.tools.Util;
Map obj = (Map) application.getAttributeValue("transientValues");
System.out.println("BEFORE RULE: Transient Values ==> " + obj);
if(null != obj) {
    String offset = obj.get("offset");
    System.out.println("BEFORE RULE: offset value ==> " + offset);
    String urlString = (String) requestEndPoint.getFullUrl();
    if(Util.isNotNullOrEmpty(offset)) {
        System.out.println("BEFORE RULE: requestEndpoint ==> " + requestEndPoint);
        System.out.println("BEFORE RULE: URL ==> " + urlString);
        URL tempUrl = new URL(urlString);
        String queryString = tempUrl.getQuery();
        System.out.println("BEFORE RULE: Query String ==> " + queryString);

        if(Util.isNotNullOrEmpty(queryString)) {
            StringBuffer queryParams = new StringBuffer();
            String[] params = tempUrl.getQuery().split("&");
            for (String param : params) {
                if(queryParams.length() > 0)
                    queryParams.append("&");
                if(param.startsWith("sysparm_offset=")) {
                    queryParams.append("sysparm_offset=");
                    queryParams.append(offset);
                } else {
                    queryParams.append(param);
                }
            }
            urlString = urlString.replace(tempUrl.getQuery(), queryParams.toString());
        }
    }
}
```

```

        System.out.println("BEFORE RULE: Updated Query String ==> " + urlString);
        requestEndPoint.setFullUrl(urlString);
    }
    System.out.println("BEFORE RULE: requestEndpoint Updated ==> " +
        requestEndPoint);
    return requestEndPoint;

```

In case of Dropbox V2, the **cursor** returned from the previous team membership listing API would be stored in the **transientValues** map in the application by the Web Service AFTER Rule. The url is modified to direct to the paging API and the cursor would be sent as a part of the form data. Ensure that the **hasMore** flag is set by the earlier requests AFTER RULE

- **Web Service After Rule:** The AFTER Rule deduces whether the managed system has more records which can be fetched and added as an entry in **transientValues** with **hasMore** key and value as TRUE/FALSE depending upon the condition deduced.

For example, see sample AFTER Rule for account aggregation request in Web Services Connector for Dropbox using V2 in **examplerules.xml** by name **Example WSAfterRI DropboxPaging** as follows:

```

Integer fetchedRecordsCount = 0;if(null != processedResponseObject) {
    fetchedRecordsCount = ((List) processedResponseObject).size();
}

Integer expectedCount = null;
Integer offset = null;
URL url = new URL(requestEndPoint.getFullUrl());
System.out.println("AFTER RULE: Original Url ==> " + url);
String[] params = url.getQuery().split("&");
for (String param : params) {
    String name = param.split("=")[0];
    String value = param.split("=")[1];

    switch(name) {
        case "sysparm_limit":
            expectedCount = Integer.parseInt(value);
            break;

        case "sysparm_offset":
            offset = Integer.parseInt(value);
            break;

        default:
    }
}

System.out.println("AFTER RULE: Fetch Count ==> " + fetchedRecordsCount);
System.out.println("AFTER RULE: Limit Count ==> " + expectedCount);
System.out.println("AFTER RULE: Fetch Offset ==> " + offset);

```

```

boolean hasMore = (fetchedRecordsCount != 0 && null != expectedCount &&
fetchedRecordsCount.equals(expectedCount) && null != offset);
System.out.println("AFTER RULE: Has More? ==> " + hasMore);

Map transientValues = application.getAttributeValue("transientValues");
if(transientValues == null) {
    transientValues = new HashMap();
    application.setAttribute("transientValues", transientValues);
}
transientValues.put("hasMore", hasMore);
if (hasMore) {
    if(null != offset) {
        System.out.println("AFTER RULE: New Offset ==> " + (offset + expectedCount));
        transientValues.put("offset", String.valueOf(offset + expectedCount));
    }
}
}

```

In case of Dropbox, Dropbox V2 for team membership response contain the following elements:

- **cursor**: is an encrypted token which represents the next page to be fetched, if any, and would form part of the subsequent API calls.
- **has_more**: is a boolean value which explicitly indicates whether more records are available for fetching.

AFTER Rule stores the **cursor** and **has_more** values from the response in the **transientValues** map in the Application object. This map stores the necessary information which would be used by the BEFORE RULE to manipulate the next API call. Ensure that the flag indicating whether the managed system contains more records is stored by the key named **hasMore**. This field is mandatory as it is the deciding factor for aborting the pagination requests.

Paging tab

Paging can be configured in Account/ Group Aggregation endpoints through Paging tab with one of the following methods:

- Paging based on limit-offset
- Paging based on response markers
- Paging based on response header links

Prior to IdentityIQ version 8.0 Patch 3, if there are multiple Account or Group Aggregation endpoints configured, paging would be supported only for the first endpoint of Account and Group Aggregation. However, for IdentityIQ version 8.0 Patch 3 onwards, paging can be configured for every child endpoint.

Paging mechanism has the following predefined set of keywords:

Keywords	Description	
application	Represents the application.	
	baseUrl	Base URL configured in the application.
endpoint	Represents endpoint configuration.	
	relativeURL	Relative URL of the endpoint.
	fullUrl	Full URL contained within the endpoint.
limit	Indicates page limit configured in the endpoint configuration.	
offset	Indicates page offset configured in the endpoint configuration.	
request	Represents request body.	
requestHeaders	Represents request header.	
response	Represents response body object.	
responseHeaders	Represents response header.	
TERMINATE_IF	Indicates termination condition, multiple conditions can exist.	
	NO_RECORDS	Indicates no records retrieved; to be used in conjunction with TERMINATE_IF, evaluates to TRUE / FALSE
	RECORDS_COUNT	Indicates number of records retrieved.
NULL	Indicates null or empty object.	
REMOVE	Indicates to remove an attribute.	

The following table lists the supported and conditional supported operations

Type	Operations
Regular operations	+, -, *, /, =, && and
Conditional operations	<, >, <=, >=, == and !=

Paging based on limit-offset

This section describes paging based on limit-offset for ServiceNow and Workday managed system.

For ServiceNow

For example, the initial aggregation url for ServiceNow managed system would be as follows: **`https://XYZ.service-now.com/api/now/v1/table/sys_user?sysparm_limit=100&sysparm_fields=sys_id`**

The above url includes the following parameters:

- baseUrl: **`https://XYZ.service-now.com`**
- relativeURL: **`api/now/v1/table/sys_user?sysparm_limit=100&sysparm_fields=sys_id`**

Following are the configuration steps in the paging tab based on limit-offset:

1. Use the following for ServiceNow managed system:

```
$sysparm_limit$ = 100  
TERMINATE_IF $RECORDS_COUNT$ < $sysparm_limit$  
$sysparm_offset$ = $sysparm_offset$ + $sysparm_limit$  
$endpoint.fullUrl$ = $application.baseUrl$ + "/api/now/v1/table/sys_  
user?sysparm_fields=sys_id&sysparm_limit=100&sysparm_offset=" + $sysparm_  
offset$
```

2. In the above example, maximum record count is been set using `sysparm_limit` and `RECORDS_COUNT`. In this case `RECORDS_COUNT` represents the number of records fetched from the response and `sysparm_limit` can be changed as required.
3. Based on the above step the next page url will be as mentioned in the steps below.
4. Second page url based on paging steps configuration: **`https://XYZ.com/api/now/v1/table/sys_user?sysparm_limit=100&sysparm_fields=sys_id&sysparm_offset=100`**
5. Third page url based on paging steps configuration: **`https://XYZ.com/api/now/v1/table/sys_user?sysparm_limit=100&sysparm_fields=sys_id&sysparm_offset=200`**

The paging would be terminated when the `RECORDS_COUNT` is less than the `sysparm_limit` as follows:
`$RECORDS_COUNT$ < $sysparm_limit$`

For Workday

For example, the initial aggregation url for Workday managed system would be as follows: **`https://XYZ.workday.com/ccx/service/sailpoint_pt1/Human_Resources/v24.1`**

The above url includes the following parameters:

- baseUrl: **`https://XYZ.workday.com`**
- relativeURL: **`/ccx/service/sailpoint_pt1/Human_Resources/v24.1`**

Following are the configuration steps in the paging tab based on limit-offset:

1. Use the following for Workday managed system:

```
TERMINATE_IF $response.wd:Response_Results.wd:Page$ > $response.wd:Response_  
Results.wd:Total_Pages$  
$offset$ = $response.wd:Response_Results.wd:Page$ + 1  
$request.bsvc:Response_Filter.bsvc:Page.text()[1]$ = $offset$
```

In the above example, the number of pages are verified with response from Workday and compared it with the total number of pages.

If the current page number is less than total pages then the request body (SOAP BODY) is updated with the new page by incrementing it.

Following is an example of initial payload request with Initial Page Size (the initial page would be the beginning index) as 1 and Page Size (number of record per page) as 10:

```
<soapenv:Body>
<bsvc:Get_Workers_Request bsvc:version="v24.1">
<bsvc:Request_Criteria>
    ....
    ....
</bsvc:Request_Criteria>
<bsvc:Response_Filter>
<bsvc:Page>1</bsvc:Page>
<bsvc:Count>10</bsvc:Count>
</bsvc:Response_Filter>
<bsvc:Response_Group>
    ....
    ....
</bsvc:Response_Group>
</bsvc:Get_Workers_Request>
</soapenv:Body>
```

Based on the above paging steps the next payload would be as follows:

2. The second aggregation payload: **Incrementing offset** (page number)

```
<soapenv:Body><bsvc:Get_Workers_Request bsvc:version="v24.1">
<bsvc:Request_Criteria>
    ....
    ....
</bsvc:Request_Criteria>
<bsvc:Response_Filter>
<bsvc:Page>2</bsvc:Page>
<bsvc:Count>10</bsvc:Count>
</bsvc:Response_Filter>
<bsvc:Response_Group>
    ....
    ....
</bsvc:Response_Group>
</bsvc:Get_Workers_Request>
</soapenv:Body>
```

Paging based on response markers

This section describes paging based on a marker value in the response. Based on the marker value in the response aggregation is terminated or continued.

The following table lists the examples of response marker based paging for the respective Managed System:

Managed System	Examples
Dropbox	<pre> TERMINATE_IF \$response.has_more\$ == FALSE \$endpoint.fullUrl\$ = \$application.baseUrl\$ + \$endpoint.relativeUrl\$ + "/continue" \$request.cursor\$ = \$response.cursor\$ REMOVE \$request.limit\$ </pre>
Salesforce	<pre> TERMINATE_IF \$response.ns:result.ns:done\$!= FALSE \$request.soapenv:Body\$ = "<urn:queryMore><urn:queryLocator>" + \$response.ns:result.ns:queryLocator\$ + "</urn:queryLocator></urn:queryMore>" </pre>
Successfactor	<pre> TERMINATE_IF \$response.ns:result.ns:hasMore\$!= TRUE \$request.soapenv:Body\$ = "<urn:queryMore><urn:querySessionId>" + \$response.ns:result.ns:querySessionId\$ + "</urn:querySessionId></urn:queryMore>" </pre>

Paging based on response header links

This section describes paging based on response header links.

The following table lists the examples of response header links based paging for the respective Managed System

Managed System	Examples
ServiceNow	<pre> TERMINATE_IF \$responseHeaders.link.next\$ == NULL \$endpoint.fullUrl\$ = \$responseHeaders.link.next\$ </pre>
Okta	<pre> TERMINATE_IF \$responseHeaders.link.next\$ == NULL \$endpoint.fullUrl\$ = \$responseHeaders.link.next\$ </pre>

Paging Configuration Caveats:

1. Every paging configuration step must start on a new line.
2. SailPoint recommends to provide a <space> after every operator, condition or placeholder for correct evaluation of paging expression.
3. Paging mechanism follows the placeholder notation for resolution of attribute values, that is., `$response.attribute_key$`. Any attribute which follows the placeholder notation would be resolved or assigned a value depending upon the operator being used.
4. Intermediate values can also be stored between page request by using the placeholder notation. In order to achieve this, any attribute key which does not match any of the predefined keywords can be used. For more information, see the example mentioned in the above table for **ServiceNow (Using limit-offset)** where

`$sysparm_offset$` is being updated and used between page requests.

- For complex expressions or conditions, multiple conditions can be clubbed together using '(' and ')'. For example:

```
TERMINATE_IF ($someattribute$ == TRUE) && ($otherattribute$ == NULL)
```

Enabling Single API Calls

The Web Services now provides support for sending data for multiple users in a single API call. If the managed system supports sending data for multiple users in single call, the single API call can be enabled/ configured by adding the **evalPattern**.

Add the following evaluation pattern function to the **Body** or **URL** to send attributes in the request:

```
evalPattern(pattern, delimiter, count)
```

The following table describes the terminologies used in the above evaluation pattern (`evalPattern`) function which is used to specify the repeated pattern of attributes data to be resolved:

Attributes	Description
pattern	Repeated pattern for set of Users in the request.
delimiter	Delimiter for the repeated patterns.
count	(Optional) Specifies the number of users data to be sent with the request endpoint. The default value is: 10 for query parameters 50 for Body parameters

For example,

- Get Users data API:
 - Required URL:** `http://Hostname:8080/identityiq/scim/v2/Users?filter=userName eq "spadmin" or userName eq "abhijit" or userName eq "wadekar" or userName eq "imran" or userName eq "test"`
 - The following Modified URL to get users data API has the `evalPattern` that would be resolved and the string would be replaced by the value of five repeated pattern of `userName` placeholders and a separator which is " or ":
 - Modified URL:** `http://Hostname:8080/identityiq/scim/v2/Users?filter=evalPattern("userName eq $response.userName$", " or ", 5)`

This functionality is not applicable for First Aggregation Endpoint.

If API supports sending limited users data then it must be specified in **evalPattern(count)** to avoid failures.

Merging of Resources would be done on ID for each endpoint which would be provided as placeholder in the pattern. For example, `$response.userName$ (userName)`.

Partitioning

The Web Services Connector supports partitioning through the following modes:

Static Partitioning: defines the separate partitions within the Application's XML through fixed mappings

Dynamic Partitioning: discovers the partitions at run time based on information returned by the remote system from a REST or Web Services API call

The Static and Dynamic partitioning modes can be used at the same time; partitions can be defined both in fixed XML configuration blocks and by dynamic values returned from the remote system. When both types of configuration are used the list of partition records are concatenated all of the defined partitions are aggregated.

Conceptually, each partition would retrieve the accounts or user records from the managed system that correspond to a specific field or field values. The connector expects that those values can be passed to a search or filter API exposed by the managed system. The connector supports partitioning through token substitution of a URL pattern and/or a POST payload body's JSON or XML.

For example, you have a Human Resources system that grouped searching for accounts by their **costCenter** attribute, and had API to return accounts like this:

GET<https://acme-hr.com/admin/listAccounts?costCenter=NAM-AdminHQ>

In this example the API would return account records belonging only to the NAM-AdminHQ costcenter.

Assuming an even distribution of accounts to various **costCenter** values, this field could be considered for partitioning up the data set. The different values for the **costCenter** attribute, in this example, could each be mapped to a Partition that aggregates the accounts corresponding to that group.

Each Partition is effectively a Map of key-value pairs that defines a unique grouping of accounts that are to be aggregated by a given partition. The partition's key/value pairs are exposed to the before/after rules of a Connector Operation. They are available in the Context URL field for appending to the request and in the body text of a POST operation.

In the Context URL the partition being aggregated is exposed as the **\$partition.*\$ keyword**, where the star is replaced by the field name from the partition to be substituted into the text. This behaves similarly to the **\$response.* key word** and other tokens used by the Web Services connector.

For example, in a partitioned account aggregation type of operation, you could have a Context URL as follows:

https://acme-hr.com/admin/listAccounts?costCenter=\$partition.costCenter\$

When evaluated during aggregation, the partition's **costCenter** field is substituted in for the **\$** sign encapsulated field and is passed to the remote system. In situations where simple path substitution is not sufficient the URL can be generated in the **WebServiceBeforeOperationRule** by modifying the **requestEndPoint's URL** field. The partition variable is exposed in the rule for use by connector-specific rule logic.

Partitioning Operations

Partitioning Aggregation is supported by the following operation types:

Get Partitions

Partitioned Account Aggregation

Administrators must configure a Partitioned Account Aggregation connector operation to support partitioning for static and dynamic. As per the name, the Partitioned Account Aggregation operation is similar to the previously existing

Account Aggregation endpoint. The only difference between the two is the Partitioned Account Aggregation operation is invoked when an Account Aggregation that has partitioning enabled has been invoked; the Partitioned Account Aggregation receives the **\$partition.\$ keyword** tokens and the original operation does not. Having these two as separate operations allows the connector to support single-threaded and partitioned (multi-threaded) aggregations with one configuration.

Administrators must configure a **Get Partition** type connector operation to support dynamic partitioning. For more information, see the following configurations.

Configuring Partitioning Aggregation

Configuring Static Partitions

Static partitions are declared inside the Application's XML file under the attribute partitions (lowercase "p"). The entry contains a list of partition records. Each partition record has a unique name and key to the value of mappings to be used inside the attributes map. The following example displays two static partitions that are identified by different **costCenter** values:

```
<entry key="partitions">
  <value>
    <List>
      <Partition name="Partition 1">
        <Attributes>
          <Map>
            <entry key="costCenter" value="CC-AA"/>
          </Map>
        </Attributes>
      </Partition>
      <Partition name="Partition 2">
        <Attributes>
          <Map>
            <entry key="costCenter" value="CC-AAZY"/>
          </Map>
        </Attributes>
      </Partition>
    </List>
  </value>
</entry>
```

Configuring Dynamic Partitioning

The Get Partitions operation returns the list of partition records from its invocation which is a **top level** operation which can be chained by referring to their Parent Operation .

For example, for a Web Service that returned the list of unique cost centers inside an HR application from an API request as follows:

GET <https://hr.acme.com/admin/listCostCenters>

```
{
  "idList": [
    "CC-00",
```

```
"CC-AA",  
"CC-AAZY",  
...  
"CC-AZYXWVUT",  
"CC-AZYXWVUTS"  
]  
}
```

In this example the JSON payload returns an `idList` that contains an array of String values, and each is a Cost Center name. In the IdentityIQ application UI a Get Partitions operation must be configured to handle this JSON response. The relevant fields would be:

- **Context URL:** /admin/listCostCenters
- **Method:** GET
- **Response - Root Path:** \$.idList.*
- **Response - Schema Attribute:** costCenter Attribute Path: \$
- **Response - Schema Attribute:** name Attribute Path: \$

By passing the string to the partition's name and `costCenter` attribute, the partition's name is rendered cleanly in the Aggregation Task's status interface. Partitions with no assigned name are randomly assigned a UUID value as their name.

When the aggregation is performed and partitioning is enabled, it would first ask the connector what dynamic partitions it has. The connector would call this Get Partitions operation and return the list of partition maps back to the aggregator process. Each partition will have a **costCenter** field and a name field populated for use in the aggregation operation.

In case of multiple endpoints, see [Configuring Multiple Get Partitions Operations](#).

Configuring Partitioned Account Aggregation Operation

The Partitioned Account Aggregation connector operation will be invoked once for each partition map. The Partitioned Account Aggregation is conceptually very similar to the regular Account Aggregation operation, and serves the same function. The only difference is the Partitioned Account Aggregation operation would always be invoked with a **\$partition.xxx\$** keyword map providing the fields that uniquely describe the partition being aggregated by the operation.

The account aggregation operations are designed to coexist. If the user selects to run a regular, non-partitioned, account aggregation then the Account Aggregation operation is used. If the user selects to run a partitioned aggregation, which is specified by which Task and Task Options are selected in IdentityIQ, then the Partitioned Account Aggregation is used. The connector supports both modes simultaneously, and professionals configuring Web Services connectors are encouraged to get a non-partitioned account aggregation functional before attempting a partitioned one. The connector logic places no restrictions or assumptions on the mode of operation; it freely supports non-partitioned, partitioned, or both at once.

The Partitioned Account Aggregation operation expects the connector administrator to substitute in one or more **\$partition.xxx\$** keyword tokens into the URL for the operation or a POST body delivered to the remote system. Continuing with the earlier example above, let's imagine we had a system that allowed us to return only the accounts or account IDs that existed in a specific Cost Center. That GET request would be displayed as follows:

GET https://hr.acme.com/admin/listAccounts?costCenter=CC-AA

```
{
  "idList": [
    "3de42cd8-d0f6-4ff6-b683-e386b246e8da",
    "73d3eacd-624e-4ca3-8b13-9095ccb9c8b3",
    ...
    "67f8ab59-2923-422d-8772-c52cb4185723"
  ]
}
```

Configuring this in a Partitioned Account Aggregation is a simple matter of passing in the **\$partition.costCenter\$** keyword into the URL, as follows:

- **Context URL:** /listAccounts?costCenter=\$partition.costCenter\$
- **Method:** GET
- **Response - Root Path:** \$.idList.*
- **Response - Schema Attribute:** costCenter Attribute Path: \$
- **Response - Schema Attribute:** name Attribute Path: \$

For example, as the non-partitioned Account Aggregation operation, the Partitioned Account Aggregation operation can be chained, with child operations referring to it by name. In the examples above the API only returns the ID property of the account record, and a subsequent GET call must be made to retrieve the full account record. The child operation or sequence of operations, that retrieve the full account, must reference the Partitioned Account Aggregation.

In case of multiple endpoints, see [Multiple Partitioned Account Aggregation](#).

Multiple Partitioned Account Aggregation

The use of multiple Partitioned Account Aggregation is allowed by the user interface. Like the Get Partitions operation, its use is intended to be strictly parent ==> child invocations. The behavior of multiple top-level **peer** Partitioned Account Aggregation operation is not recommended.

Configuring Multiple Get Partitions Operations

The user interface of IdentityIQ allows specifying multiple Get Partitions operations in the interface. For **parent ==>child** relationships, supporting when multiple HTTP calls must be made to build the internal content of Partition maps. The user interface, however, does not require the **parent ==> child** relationship to be specified and this could be the cause of some confusion during application configuration.

It is not recommended to use multiple **peer** Get Partitions operations. Ensure that this configuration returns partition maps containing consistent fields produced by the different operations.

For example,

```
{
  "name": "Department 417",
```

```
"departmentNumber": "D-417"
}
{
"name": "Cost Center CC-12",
"costCenter": "CC-12"
}
```

These two types of Partition maps is discouraged because there is no mechanism to map these different types of partitions over to Partition Account Aggregation operations that handle the presence or absence of specific fields in the partition's map. Yes, this could be handled in a Before Operation rule that modify the URL.

Exploiting the rule to that level of complexity is certainly possible, but is outside the scope of the design of partitioning support.

If the same partition fields are there in static and dynamic then the data would be fetched and aggregated twice. This is considered an aberrant configuration. In this case connector currently does not have the capability to display warnings on or removing duplicate partitions.

If static and dynamic are set to return different fields, like **costCentre** and **userType** then the connector must use a Before Operation rule to handle the URL substitution and/or in the POST body generation.

Configuring Multiple Entitlement Requests

To enable the functionality of sending multiple entitlement request of different type of entitlements (role, permission, groups and so on) in a single request to the managed system, set the value of **addRemoveEntInSingleReq** parameter to true as follows:

```
<entry key="addRemoveEntInSingleReq">
  <value>
    <Boolean>true</Boolean>
  </value>
</entry>
```

1. If **addRemoveEntInSingleReq** is set to true, then the payload for entitlements must be as given in the following example:

```
{
  "group_id" : $plan.groups$,
  "permission":$plan.permission$,
  "roles": $plan.roles$
}
```

2. If **addRemoveEntInSingleReq** is set to false, then the payload for entitlements must be as given in the following example:

```
{  
  "group_id" : "$plan.groups$",  
  "permission": "$plan.permission$",  
  "roles": "$plan.roles$"  
}
```


Schema Attributes

Discover schema functionality is not available. Hence user must add the schema attributes manually for the respective Web Service based managed system.

Create New Group

Perform the following to create new group:

1. Click on **Add Object Type** button and provide the name for the group object. For example, Role
2. This will add new schema for the newly added group object type.
3. Add the schema attributes with appropriate type in newly added group schema.
4. Provide the Native Object type, Identity attribute and display attribute for the Group Object.
5. Add new attribute in account/group schema and select the type as newly added group object.

The value of this attribute must be same as the Identity attribute value of the newly added group object.

Web Services Before/After Operation Rule

Web Services Connector uses the following operation rules:

- WebServiceBeforeOperationRule
- WebServiceAfterOperationRule

Following is the standard template used by Web Services Connector for before/after operation rules:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Rule PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Rule name="Example Rule" type="WebServiceBeforeOperationRule/
WebServiceAfterOperationRule">
  <Description>Describe your rule here.</Description>
  <Source><![CDATA[
// Add your logic here.
]]></Source>
</Rule>
```

WebServiceBeforeOperationRule

This rule is used to calculate attributes before a web-service operation call. This rule is used by the Web Services Connector before performing any operation like test connection, aggregation and so on.

Input Objects

The following table lists the input Objects for `WebServiceBeforeOperationRule`:

Objects	Type	Description
application	sailpoint.object.Application	The application whose data file is being processed. The application whose data file is being processed.
requestEndPoint	sailpoint.connector.webservices.EndPoint	Current request information; contains the header, body, context url, method type, response attribute map, successful response code
oldResponseMap	java.util.Map	The response object returned from earlier endpoint configuration of same operation type like Account Aggregation, Get Object and so on.
restClient	sailpoint.connector.webservices.WebServicesClient	This is a WebServicesClient (HttpClient) object that would enable the user to call Web Services API to target system.
provisioningPlan	sailpoint.object.ProvisioningPlan	Used to update the payload of the http request. Provisioning plan has an account request which

Objects	Type	Description
		defines the operation to be performed on the account. An account request can contain multiple attributes requests and each attribute request represents an operation on a single account attribute. This argument enables the user to update the body/payload or URL attributes of endpoint object using the provisioningPlan information.

Return Objects

The following table lists the return objects for WebServiceBeforeOperationRule:

Type	Purpose
sailpoint.connector.webservices.Endpoint/ sailpoint.connector.webservices.Map	The rule allows user to return the Endpoint object (requestEndPoint) or a map. The map can hold updatedEndPoint and connectorStateMap keys where the value expected is Endpoint (requestEndPoint) and connectorStateMap object respectively. The connectorStateMap object is a map that contains key and value of the attribute that must be updated in the application through rule.

Examples for Before Operation Rule

- Following example displays the rule used by the Web Services Connector before performing any operation like Test Connection, Aggregation and so on.

```
<?xml version='1.0' encoding='UTF-8'?> <!DOCTYPE Rule PUBLIC "sailpoint.dtd"
"sailpoint.dtd"> <Rule name="Example Rule"
type="WebServiceBeforeOperationRule"> <Description> This rule is used by the
Web Services connector before performing any operation like testconnection,
aggregation etc. </Description> <Source><![CDATA[ import
com.google.gson.Gson; import sailpoint.tools.Util; Map obj = (Map)
application.getAttributeValue("transientValues"); if(null != obj) { String
cursor = obj.get("cursor"); if(Util.isNotNullOrEmpty(cursor)) { Map
postBodyMap = (Map) requestEndPoint.getBody(); postBodyMap.put("jsonBody", "
{\"cursor\":\"\" + cursor + "\"}"); requestEndPoint.setFullUrl
(requestEndPoint.getFullUrl() + "/continue"); } } return requestEndPoint;
]]></Source> </Rule>
```

- Following example is for provisioningPlan rule:

```
import java.util.ArrayList; import java.util.HashMap; import java.util.Map;
import connector.common.JsonUtil; import connector.common.Util; import
sailpoint.connector.webservices.Endpoint; import
sailpoint.connector.webservices.WebServicesClient; import
sailpoint.object.Application; import sailpoint.object.ProvisioningPlan;
import sailpoint.object.ProvisioningPlan.AccountRequest; Map body =
requestEndPoint.getBody(); String jsonBody = (String) body.get("jsonBody");
log.info("Rule - Modify Body: running"); try { Map jsonMap = JsonUtil.toMap
```

```

(jsonBody); if (jsonMap != null) { Object roleEntry = jsonMap.get
("webSiteAndRole"); String role = ""; if (roleEntry != null && roleEntry
instanceof ArrayList) { ArrayList rolesArray = (ArrayList) roleEntry; if
(rolesArray.size() > 0) { role = (String) rolesArray.get(0); } } else if
(roleEntry != null) { role = (String) roleEntry; } jsonMap.remove
("webSiteAndRole"); jsonMap.put("webSiteAndRole", role); log.info("Rule -
Modify Body: setting webSiteAndRole = " + role); String webID = ""; if
(provisioningPlan != null) { log.info("Rule - Modify Body: plan is not
null"); for (AccountRequest accReq : Util.iterate
(provisioningPlan.getAccountRequests())) { log.info("Rule - Modify Body:
iterating over account requests"); for (ProvisioningPlan.AttributeRequest
attReq : Util.iterate(accReq.getAttributeRequests())) { log.info("Rule -
Modify Body: iterating over attribute requests"); String attrName =
attReq.getName(); String value = null; if (attrName != null &&
"webID".equalsIgnoreCase(attrName)) { webID = (String) attReq.getValue();
log.info("Rule - Modify Body: setting webID = " + webID); } } } } else {
log.info("Rule - Modify Body: plan is null"); } if (!"".equals(webID)) {
jsonMap.put("webID", webID); } // Add in any other missing fields that are
required if (!jsonMap.containsKey("webLogonEmail")) { jsonMap.put
("webLogonEmail", ""); } if (!jsonMap.containsKey("taxID")) { jsonMap.put
("taxID", ""); } if (!jsonMap.containsKey("taxIdType")) { jsonMap.put
("taxIdType", ""); } if (!jsonMap.containsKey("actorLogonId")) { jsonMap.put
("actorLogonId", ""); } String finalBody = JsonUtil.render(jsonMap);
body.put("jsonBody", finalBody); requestEndPoint.setBody(body); } } catch
(Exception ex) { log.error("Rule - Modify Body: " + ex); } return
requestEndPoint;

```

- Following is an example of Web Services Connector for executing before operation rule on target system API:

```

import java.util.*; import connector.common.JsonUtil; import
sailpoint.connector.webservices.WebServicesClient; try { WebServicesClient
client = new WebServicesClient(); String url =
"http://hostname:portnumber/RestFormDataWS/beforeRuleMock"; Map args = new
HashMap(); args.put(WebServicesClient.ARG_URL, url); client.configure(args);
Map header = new HashMap(); // Information can be fetched from
requestEndpoint and updated in the header and body header.put
("Authorization", "XXXX XXXX"); header.put("Content-
Type", "application/json"); List<String> allowedStatuses = new ArrayList();
allowedStatuses.add("2**"); Map payload = new HashMap(); payload.put
("jsonBody", "{ \"user\":
{ \"userId\": \"RinkuSharma\", \"firstName\": \"Rinku\", \"lastName\": \"Sharma\", \"title\": \"man
ager\" } }"); String response = client.executePost(url, payload, header,
allowedStatuses); // if response contains token it can be updated in the
requestEndpoint header or body // the requestEndpoint will be used for
execution of the particular operation configured log.info("response: " +
response); } catch (Exception e) { log.error(e.getMessage(), e); }

```

- Following example displays before operation rule that can be used to update the value in endpoint and application attributes value before execution of the endpoint:

```

Map updatedInfoMap = new HashMap(); requestEndPoint.setFullUrl
(requestEndPoint.getFullUrl().replaceAll("&&", "&")); Map connectorStateMap
= new HashMap(); connectorStateMap.put("accesstoken", "Bearer
accessTokenGeneratedInBeforeRuleScript"); updatedInfoMap.put

```

```
("updatedEndPoint", requestEndPoint); updatedInfoMap.put
("connectorStateMap", connectorStateMap); return updatedInfoMap;
```

WebServicesAfterOperationRule

This rule is used to calculate attributes after a web-service operation call.

This rule is used by the Web Services Connector to update parsed resource object and save **connectorStateMap** values. Create List of Objects would later be converted to Resource object and save **connectorStateMap** values to the application object permanently.

The returned map will hold data key for resource object list and **connectorStateMap** key for updated attribute that must be saved in the application.

Input Objects

Objects	Type	Description
application>	sailpoint.object.Application	Application whose data file is being processed.
requestEndPoint	sailpoint.connector.webservices.EndPoint	Current request information; contains the header, body, context url, method type, response attribute map, successful response code.
processedResponseObject	List<Map<String, Object>>	This object is List of Map (account/group). The map contains key as identityAttribute of the application schema and value is all the account/group attributes (schema) passed by the connector after parsing the respective API response.
rawResponseObject	String	String object which holds the raw response returned from the target system which can be in JSON or XML form.
restClient	sailpoint.connector.webservices.WebServicesClient	A WebServicesClient (HttpClient) object that would enable the user to call the Web Services API target system.

Return Objects

Type	Purpose
java.util.Map	<p>The Map object returned from the After Operation Rule may contain any or all of the following:</p> <ul style="list-style-type: none"> Updated list of account / group resource objects; identified by key data Attribute values to be updated into application via connector state map; identified by key connectorStateMap <p>Each resource (account/group) object is of type Map which contains key-value pair, where key represents the schema attribute name and value represents the account/ group attribute value.</p>

Examples for After Operation Rules

Following example displays after operation rule that can be used to update the accounts/groups and application attributes value after execution of the endpoint:

```
import java.util.*; import java.util.HashMap; import java.util.List; import
java.util.Map; import java.util.Map.Entry; import java.util.Iterator; //We can
parse the response(rawResponseObject) to fetch from the respective executed
Endpoint operation. Map updatedMapInfo = new HashMap(); if
(processedResponseObject != null){ for (Map iterateMap : processedResponseObject)
{ if (iterateMap != null ) { Set keySet = iterateMap.keySet(); for (String s :
keySet) { if (s.equals("given_name")) { String forStr = (String) iterateMap.get
("given_name"); forStr = "TEST"+ forStr; iterateMap.put("given_name", forStr); }
} } } updatedMapInfo.put("data", processedResponseObject); } Map
connectorStateMap = new HashMap(); connectorStateMap.put("refresh_
token","refreshTokenGeneratedInAfterRuleScript"); updatedInfoMap.put
("connectorStateMap",connectorStateMap); return updatedMapInfo;
```

Following example displays the after operation rule that can be used in Web Services connector for updating the resource object:

```
<![CDATA[ import connector.common.JsonUtil; import java.util.HashMap; import
java.util.Map.Entry; import java.util.Map; import java.util.List; import
java.util.ArrayList; import javax.net.ssl.HttpURLConnection; import
java.io.BufferedReader; import java.io.PrintStream; import java.io.StringWriter;
import java.text.SimpleDateFormat; import sailpoint.tools.GeneralException; Map
updatedMapInfo = new HashMap(); List list = new ArrayList(); ArrayList<String>
Roles = new ArrayList<String>(); Map response = (Map) JsonUtil.toMap
(rawResponseObject); int RoleSize = 0; String newName; List Finallist = new
ArrayList(); List workspace = new ArrayList(); log.error("RULEWS response at
start" + response); if (response.get("data") != null) { list = (ArrayList)
response.get("data"); for(int d = 0; d < list.size(); d++ ){ Map responseMap =
(Map) list.get(d); if (responseMap.get("attributes") != null) { Map newmap = new
HashMap(); Map data = (Map) responseMap.get("attributes"); newmap.put
("firstName", data.get("firstName")); newmap.put("lastName",data.get
("lastName")); newmap.put("displayName",data.get("displayName")); newmap.put
("userName",data.get("userName")); newmap.put("email",data.get("email")); if
(data.get("workspaceMemberships") != null) { ArrayList Workspacedetail =
(ArrayList) data.get("workspaceMemberships"); for (int i = 0; i <
Workspacedetail.size(); i++) { Map work = (Map) Workspacedetail.get(i); for (int
w = 0; w < work.size(); w++) { if (work.get("workspaceName") != null) {
workspace.add(work.get("workspaceName")); Roles = (ArrayList) work.get
("workspaceRoles"); for (int r = 0; r < Roles.size(); r++) { if (Roles.get(r) !=
null) { newName = Roles.get(r).toString() + " - " + work.get("workspaceName"); if
(newName != null) { Roles.set(r, newName); newmap.put("workspaceRoles", Roles); }
} } } break; } } } Finallist.add(newmap); } } } log.error("RULEWS newmap at end"
+ newmap); log.error("RULEWS Finallist at end" + Finallist); log.error("RULEWS
processedResponseObject Before is " + processedResponseObject);
updatedMapInfo.put("data", Finallist); log.error("RULEWS updatedMapInfo is " +
updatedMapInfo); return updatedMapInfo; log.error("RULEWS processedResponseObject
after is " + processedResponseObject); ]]
```

Web Services Class used in Before/After Operation Rule

This section describes the following types of Web Services class:

- WebServicesClient
- EndPoint Class

The Web Services Classes mentioned in this section are general guidelines. New classes/methods would be added.

WebServicesClient

The following table list the different rules and their description:

Rule	Description
Constructor Detail	
WebServicesClient	Default constructor.
	<code>public WebServicesClient()</code>
	Constructor that configures the client using the given args.
	<code>public WebServicesClient(java.util.Map args) throws java.lang.Exception</code> Throws: <code>java.lang.Exception</code>
Method Detail	
configure	Configure connection parameters. See the ARG_* constants. <code>public void configure(java.util.Map args) throws java.lang.Exception</code> Throws: <code>java.lang.Exception</code>
executeGet	<ul style="list-style-type: none"> • Execute method GET with headers <code>public java.lang.String executeGet(java.util.Map headers, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</code> Parameters: headers (Request headers) and allowedStatuses (Allowed status codes) • Execute method GET with URL and headers <code>public java.lang.String executeGet(java.lang.String url, java.util.Map headers, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</code> Parameters: url (Request URL), headers (Request headers) and allowedStatuses (Allowed status codes) Returns: Response object Throws: <code>java.lang.Exception</code>
executePost	<ul style="list-style-type: none"> • Execute method POST with URL, payload and headers <code>public java.lang.String executePost(java.lang.String url,</code>

Rule	Description
	<pre>java.lang.Object payload, java.util.Map headers, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</pre> <p>Parameters: url (Request URL), payload (Request body), headers (Request headers) and allowedStatuses (Allowed status codes)</p> <ul style="list-style-type: none"> • Execute method POST with URL and payload <pre>public java.lang.String executePost(java.lang.String url, java.lang.Object payload, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</pre> <p>Parameters: url (Request URL), payload (Request body) and allowedStatuses (Allowed status codes)</p> <p>Returns: Response object</p> <p>Throws: java.lang.Exception</p>
executePut	<ul style="list-style-type: none"> • Execute method PUT with URL and payload <pre>public java.lang.String executePut(java.lang.String url, java.lang.Object payload, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</pre> <p>Parameters: url (Request URL), payload (Request body) and allowedStatuses (Allowed status codes)</p> • Execute method PUT with URL, payload and headers <pre>public java.lang.String executePut(java.lang.String url, java.lang.Object payload, java.util.Map headers, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</pre> <p>Parameters: url (Request URL), payload (Request body), headers (Request headers) and allowedStatuses (Allowed status codes)</p> <p>Returns: Response object</p> <p>Throws: java.lang.Exception</p>
executePatch	<ul style="list-style-type: none"> • Execute method PATCH with URL and payload <pre>public java.lang.String executePatch(java.lang.String url, java.lang.Object payload, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</pre> <p>Parameters: url (Request URL), payload (Request body) and allowedStatuses (Allowed status codes)</p> • Execute method PATCH with URL, payload and headers <pre>public java.lang.String executePatch(java.lang.String url, java.lang.Object payload, java.util.Map headers, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</pre> <p>Parameters: url (Request URL), payload (Request body), headers (Request headers) and allowedStatuses (Allowed status codes)</p> <p>Returns: Response object</p>

Rule	Description
	Throws: <code>java.lang.Exception</code>
<code>getResponseHeaders</code>	Get last executed Request's Response headers. <code>public java.util.Map<java.lang.String,java.lang.String> getResponseHeaders ()</code>
<code>executeDelete</code>	<ul style="list-style-type: none"> Execute method DELETE with URL <code>public java.lang.String executeDelete(java.lang.String url, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</code> Parameters: url (Request URL) and allowedStatuses (Allowed status codes) Execute method DELETE with URL and headers <code>public java.lang.String executeDelete(java.lang.String url, java.util.Map headers, java.util.List<java.lang.String> allowedStatuses) throws java.lang.Exception</code> Parameters: url (Request URL), headers (Request headers) and allowedStatuses (Allowed status codes) Returns: Response object Throws: <code>java.lang.Exception</code>

EndPoint Class

The following table list the different rules and their description:

Rule	Description
Constructor Detail	
<code>EndPoint</code>	<code>public EndPoint()</code>
Method Detail	
<code>setAfterRule</code>	Setting the after rule name <code>public void setAfterRule(java.lang.String value)</code>
<code>setBeforeRule</code>	Setting the before rule name <code>public void setBeforeRule(java.lang.String value)</code>
<code>setParseRule</code>	<code>public void setParseRule(java.lang.String value)</code>
<code>setContextUrl</code>	Set the context url for the particular operation (create user, update user, account aggregation, and so on) <code>public void setContextUrl(java.lang.String value)</code>
<code>setHttpMethodType</code>	Set the http method (put, post, get, patch and delete) for the particular operation (create user, update user, account aggregation, and so on) <code>public void setHttpMethodType(java.lang.String value)</code>

Rule	Description
setOperationType	<p>Set the operation (Account Aggregation, Group Aggregation, Create Account etc) for the particular operation record (create user, update user, account aggregation, and so on)</p> <pre>public void setOperationType(java.lang.String value)</pre>
setRootPath	<p>Set the root of the JSON response returned from the managed system (Managed system) for the particular operation (create user, update user, account aggregation, and so on)</p> <pre>public void setRootPath(java.lang.String value)</pre>
setFullUrl	<p>set the complete url (endpoint) of the operation that need to be performed for the particular operation (create user, update user, account aggregation, and so on)</p> <pre>public void setFullUrl(java.lang.String value)</pre>
setBaseUrl	<p>Set the base url (the machine id or IP and the port where the service is executing) for the operation that need to be performed. Ideally this would be common for all the operation.</p> <pre>public void setBaseUrl(java.lang.String value)</pre>
setSequenceNumberForEndpoint	<p>Set the Sequence number particular operation (create user, update user, account aggregation, and so on)</p> <pre>public void setSequenceNumberForEndpoint(int value)</pre>
setUniqueNameForEndPoint	<p>Set Unique operation name for particular operation (create user, update user, account aggregation, and so on)</p> <pre>public void setUniqueNameForEndPoint(java.lang.String value)</pre>
setResMappingObj	<p>Set the Response mapping for the response attribute returned in the JSON response from the managed system (Managed system) for the particular operation like create user, update user, account aggregation, and so on. Here the key would be attribute name (attribute in the schema) and value would be the JSON response path after the root path mentioned above.</p> <pre>public void setResMappingObj(java.util.Map value)</pre>
setHeader	<p>Set HTTP header information in the form of key value (For example, key='ContentType' Value='Application/JSON')</p> <pre>public void setHeader(java.util.Map value)</pre>
addHeader	<p>Adding key value if header exists or will create header and add</p> <pre>public void addHeader(java.lang.String entry, java.lang.String value)</pre>
setBody	<p>Set http body information as a Map. Here the map would contain three keys</p>

Rule	Description
	<p>bodyFormat, bodyFormData and jsonBody. The bodyFormat value can be raw that means user has provided values as raw JSON string else user has provided value in the key value format that must be converted into JSON.</p> <pre>public void setBody(java.util.Map value)</pre>
setResponseCode	<p>Set the value of successful response code as list (200, 299, 201). This would be respected by the connector if any other response code would be consider as request failure.</p> <pre>public void setResponseCode(java.util.List value)</pre>
getAfterRule	<p>Fetch the name of after rule assigned to the particular operation like create, update user, account aggregation, and so on:</p> <pre>public java.lang.String getAfterRule()</pre>
getBeforeRule	<p>Fetch the name of before rule assigned to the particular operation like create, update user, account aggregation, and so on.</p> <pre>public java.lang.String getBeforeRule()</pre>
getParseRule	<pre>public java.lang.String getParseRule()</pre>
getContextUrl	<p>Fetch the contextUrl provided to the particular operation like create, update user, account aggregation, and so on</p> <pre>public java.lang.String getContextUrl()</pre>
getHttpMethodType	<p>Fetch the httpMethodType (get,put,post,delete and patch) provided to the particular operation like create, update user, account aggregation, and so on.</p> <pre>public java.lang.String getHttpMethodType()</pre>
getOperationType	<p>Fetch the operationType (Account Aggregation,Create Account,Group Aggregation etc) provided to the particular operation like Create, update user, account aggregation, and so on.</p> <pre>public java.lang.String getOperationType()</pre>
getRootPath	<p>Fetch the rootPath provided to the particular operation like Create, update user, account aggregation, and so on.</p> <pre>public java.lang.String getRootPath()</pre>
getFullUrl	<p>Fetch the fullUrl that is a combination of basicUrl + contextUrl for the particular operation like Create, update user, account aggregation, and so on.</p> <pre>public java.lang.String getFullUrl()</pre>
getBaseUrl	<p>Fetch the baseUrl which is common for all operation like Create, update user, account aggregation, and so on.</p> <pre>public java.lang.String getBaseUrl()</pre>

Rule	Description
getSequenceNumberForEndpoint	Fetch the sequenceNumber for the particular operation (Create, update user, account aggregation, etc) that decide the priority of execution for operation, if there are multiple endpoint of same operation like account aggregation. public int getSequenceNumberForEndpoint()
getUniqueNameForEndPoint	Fetch the uniqueName provided to the particular operation like Create, update user, account aggregation, and so on. public java.lang.String getUniqueNameForEndPoint()
getResMappingObj	Fetch the responseMapping map that will have key as schema attribute and value as JSON path in the JSON response for particular operation like Create, update user, account aggregation, and so on. public java.util.Map getResMappingObj()
getHeader	Fetch the Http header map that holds the header information for particular operation like Create, update user, account aggregation, and so on. public java.util.Map getHeader()
getBody	Fetch the body map that holds the body information with keys like bodyFormat , jsonBody and bodyFormData . The bodyFormat can have raw or formData value. bodyFormData will have value as map jsonBody will have value as string with whole JSON. public java.util.Map getBody()
getResponseCode	Fetch the success response code (list) value which will decide whether the operation was successful or not for particular operation like Create, update user, account aggregation, and so on. public java.util.List getResponseCode()
getAttributes	public sailpoint.object.Attributes getAttributes()
getAttribute	public java.lang.Object getAttribute(java.lang.String name)
getBooleanAttributeValue	public boolean getBooleanAttributeValue(java.lang.String name)
getStringAttributeValue	public java.lang.String getStringAttributeValue(java.lang.String name)
setAttribute	public void setAttribute(java.lang.String name, java.lang.Object value)
getPaginationSteps	Fetch the paging steps as a string which will decide how account/group paging will work. public java.lang.String getPaginationSteps()
setPaginationSteps	Set the paging steps as a string which will decide how account/group paging

Rule	Description
	would work. <pre>public void setPaginationSteps(java.lang.String paginationSteps)</pre>
toString	<pre>public java.lang.String toString()</pre> Overrides: toString in class java.lang.Object
getResponseBody	Retrieve last executed Request's Response Body. <pre>public java.lang.String getResponseBody()</pre>
setXpathNamespaces	Sets XPath namespaces using the supplied Map object. <pre>public void setXpathNamespaces(Map<String, String> xpathNamespaces)</pre>
getXpathNamespaces	Retrieves XPath namespaces. <pre>public Map<String, String> getXpathNamespaces()</pre>
getPagingInitialOffset()	Retrieves the initial page offset. <pre>public int getPagingInitialOffset()</pre>
setPagingInitialOffset()	Sets the initial paging offset. <pre>public void setPagingInitialOffset(int pagingInitialOffset)</pre>
getPagingSize()	Retrieves the page limit. <pre>public int getPagingSize()</pre>
setPagingSize	Sets the page limit. <pre>public void setPagingSize(int pagingLimit)</pre>

Troubleshooting

1 - TLS communication failure for Web Services Connector when IdentityIQ is deployed on IBM WebSphere Application Server

TLS certificates can be imported and configured in WebSphere's trust stores through **SSL certificate and key management ==> Key stores and certificates ==> CellDefaultTrustStore**.

However, the Web Services Connector depends on the trust store of the JDK during TLS communication and does not respect the WebSphere's trust stores. This results in TLS communication failure.

Resolution: An additional configuration is required to override and set the default trust store as WebSphere's trust store. Perform the following:

1. Select **Servers ==> Application Servers ==> _server_name_ ==> Process Definition ==> Java Virtual Machine ==> Custom Properties ==> New**
2. Add the following properties:
 - Property Name: `javax.net.ssl.trustStore`
Value: `<WebSphere truststore file path>`
 - Property Name: `javax.net.ssl.trustStorePassword`
Value: `<Truststore Password>`

2 - Test Connection fails when Web Services is using TLS 1.0 on WebSphere Application Server

With this release Web Services Connector is enhanced to enforce the secure communication. Hence this may cause **Test Connection** to fail with the following error if IBM JDK 1.8 is used:

```
[ConnectorException] [Possible suggestions] Ensure configuration parameters are correct with a valid format, Ensure active network connectivity between Source and Target system. [Error details] javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure
```

Resolution: Perform the following to add the `com.ibm.jsse2.overrideDefaultTLS` property in the Java properties:

1. Navigate to **Servers ==> Java and Process Management ==> Process Definition ==> JVM ==> Custom?Properties** and add the following property:
`com.ibm.jsse2.overrideDefaultTLS`
OR
Add in `jvm.options` file. For more information, see the following the link:
https://www.ibm.com/support/knowledgecenter/SSD28V_liberty/com.ibm.websphere.wlp.core.doc/a e/t-wlp_admin_customvars.html
2. Restart the server after changes.

3 - An issue occurs during aggregation when attribute in JSON response has (.) dot character in the name

Resolution: If attribute in JSON response has (.) dot character, for example, `"Attribute.Name"`, then set the attribute path value as `["Attribute.Name"]`.

For example,

```
{  
  "result": {  
    "manager.user_name": "Test"  
  }  
}  
JSON Path : $.result["manager.user_name"]
```