NetIQ® Identity Manager

Driver for REST Implementation Guide

November 2022

Legal Notice

Contents

About NetIQ Corporation

We are a global, enterprise software company, with a focus on the three persistent challenges in your environment: Change, complexity and risk—and how we can help you control them.

Our Viewpoint

Adapting to change and managing complexity and risk are nothing new

In fact, of all the challenges you face, these are perhaps the most prominent variables that deny you the control you need to securely measure, monitor, and manage your physical, virtual, and cloud computing environments.

Enabling critical business services, better and faster

We believe that providing as much control as possible to IT organizations is the only way to enable timelier and cost effective delivery of services. Persistent pressures like change and complexity will only continue to increase as organizations continue to change and the technologies needed to manage them become inherently more complex.

Our Philosophy

Selling intelligent solutions, not just software

In order to provide reliable control, we first make sure we understand the real-world scenarios in which IT organizations like yours operate—day in and day out. That's the only way we can develop practical, intelligent IT solutions that successfully yield proven, measurable results. And that's so much more rewarding than simply selling software.

Driving your success is our passion

We place your success at the heart of how we do business. From product inception to deployment, we understand that you need IT solutions that work well and integrate seamlessly with your existing investments; you need ongoing support and training post-deployment; and you need someone that is truly easy to work with—for a change. Ultimately, when you succeed, we all succeed.

Our Solutions

- Identity & Access Governance
- Access Management
- Security Management
- Systems & Application Management
- Workload Management
- Service Management

Contacting Sales Support

For questions about products, pricing, and capabilities, contact your local partner. If you cannot contact your partner, contact our Sales Support team.

Worldwide:	www.netiq.com/about_netiq/officelocations.asp
United States and Canada:	1-888-323-6768
Email:	info@netiq.com
Web Site:	www.netiq.com

Contacting Technical Support

For specific product issues, contact our Technical Support team.

Worldwide:	www.netiq.com/support/contactinfo.asp
North and South America:	1-713-418-5555
Europe, Middle East, and Africa:	+353 (0) 91-782 677
Email:	support@netiq.com
Web Site:	www.netiq.com/support

Contacting Documentation Support

Our goal is to provide documentation that meets your needs. The documentation for this product is available on the NetIQ Web site in HTML and PDF formats on a page that does not require you to log in. If you have suggestions for documentation improvements, click **Add Comment** at the bottom of any page in the HTML version of the documentation posted at www.netiq.com/documentation. You can also email Documentation-Feedback@netiq.com. We value your input and look forward to hearing from you.

Contacting the Online User Community

NetIQ Communities, the NetIQ online community, is a collaborative network connecting you to your peers and NetIQ experts. By providing more immediate information, useful links to helpful resources, and access to NetIQ experts, NetIQ Communities helps ensure you are mastering the knowledge you need to realize the full potential of IT investments upon which you rely. For more information, visit community.netiq.com.

About This Guide

This guide explains how to install and configure the Identity Manager Driver for REST to establish communication between the Identity Manager and the connected application. The guide includes the following information:

- Chapter 1, “Understanding the REST Driver,” on page 13
- Chapter 2, “Installing the Driver Files,” on page 27
- Chapter 3, “Creating A New Driver Object,” on page 29
- Chapter 4, “Upgrading an Existing Driver,” on page 47
- Chapter 5, “Customizing the Driver for RESTful Services,” on page 51
- Chapter 6, “Securing Communication,” on page 65
- Chapter 7, “Managing the Driver,” on page 69
- Chapter 8, “Use Case Based Deployment of REST Driver with Connected Applications,” on page 71
- Chapter 9, “Troubleshooting the Driver,” on page 91
- Appendix A, “Driver Properties,” on page 93
- Appendix B, “Using Java Extensions,” on page 105
- Appendix C, “Trace Levels,” on page 109
- Appendix D, “Supported JSON Format,” on page 111

Audience

This guide is intended for administrators implementing Identity Manager, application server developers, Web services administrators, and consultants. You should also have an understanding of DSML/SPML, REST, JSON, and HTML.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation, or go to www.novell.com/documentation/feedback.html and enter your comments there.

Documentation Updates

For more information about the library for Identity Manager, see the following resources:

- Identity Manager documentation website (https://www.netiq.com/documentation/identity-manager-48/)
- Identity Manager drivers documentation website (https://www.netiq.com/documentation/identity-manager-48-drivers/)

About this Book and the Library

The *Identity Manager Driver for REST Implementation Guide* explains how to install and configure the Identity Manager Driver for REST.

Intended Audience

This book provides information for administrators implementing Identity Manager, application server developers, Web services administrators, and consultants, who also have an understanding of DSML/SPML, REST, JSON and HTML.

Other Information in the Library

For more information about the library for Identity Manager, see the following resources:

- Identity Manager documentation website (https://www.netiq.com/documentation/identity-manager-48/)
- Identity Manager drivers documentation website (https://www.netiq.com/documentation/identity-manager-48-drivers/)

1 Understanding the REST Driver

REST (**Representational State Transfer**) is an HTTP-based protocol used for Internet communication. REST is the widely emerging standard for applications across World Wide Web, Software as a Service (SaaS) applications, distributed systems, cloud-based services, web services and other business critical applications. A RESTful service is implemented using the HTTP protocol and the principles of REST.

The Identity Manager driver for REST enables identity provisioning and data synchronization between an Identity Vault and any RESTful service.

The driver is not targeted to a specific Web service. The driver is a generic shim that handles the HTTP transport of data between an Identity Vault and a RESTful service. For this driver, a RESTful service is defined as an application that uses HTTP as the transport protocol. The REST driver provides interfaces to transform events and data between Identity Vault and connected system. The driver also exposes REST endpoints that enables Identity Manager to function as a RESTful service.

The driver provides the following key features:

- Supports Anonymous, Basic, and OAuth2.0 authentication
- Supports XML/JSON based requests between the Identity Manager and any RESTful services
- Provides interfaces to extend driver functionalities
- Exposes the REST endpoints that enables CRUD operation to be done in RESTful way on Identity Vault
- Supports password synchronization

This section provides the following information for the REST driver:

- "Key Terms" on page 13
- "Driver Concepts" on page 15
- "Support for Standard Driver Features" on page 21

Key Terms

- "Identity Manager" on page 14
- "Connected System" on page 14
- "Identity Vault" on page 14
- "Identity Manager Engine" on page 14
- "Driver Shim" on page 14
- "Driver Packages" on page 14
- "Remote Loader" on page 15

Identity Manager

NetIQ Identity Manager is a service that synchronizes data among servers in a set of connected systems by using a robust set of configurable policies. Identity Manager uses the Identity Vault to store shared information, and uses the Identity Manager engine for policy-based management of the information as it changes in the vault or connected system. Identity Manager runs on the server where the Identity Vault and the Identity Manager engine are located.

Connected System

A connected system is any system that can share data with Identity Manager through a driver. Any RESTful service is a connected system for this driver.

Identity Vault

The Identity Vault is a persistent database powered by eDirectory and used by Identity Manager to hold data for synchronization with a connected system. The vault can be viewed narrowly as a private data store for Identity Manager or more broadly as a metadirectory that holds enterprise-wide data. Data in the vault is available to any protocol supported by eDirectory, including the NetWare Core Protocol (NCP), which is the traditional protocol used by Identity Console, and LDAP.

Because the vault is powered by eDirectory, Identity Manager can be easily integrated into your corporate directory infrastructure by using your existing directory tree as the vault.

Identity Manager Engine

The Identity Manager engine is the core server that implements the event management and policies of Identity Manager. The engine runs on the Java Virtual Machine in eDirectory.

Driver Shim

A driver shim is the component of a driver that converts the XML-based Identity Manager command and event language (XDS) to the protocols and API calls needed to interact with a connected system. The shim is called to execute commands on the connected system after the Output Transformation runs. Commands are usually generated on the Subscriber channel but can be generated by command write-back on the Publisher channel.

Driver Packages

The REST driver packages are available on the Package Update site. When you create a driver with packages in Designer, Designer creates a set of policies and rules suitable for synchronizing with the REST driver.

The REST driver packages are:

- **NETQRESTBASE**: A mandatory package for the REST driver containing basic driver settings with handlers, to establish connection with the connected application.
- **NETQRESTDCFG**: An optional package with some basic default configuration, can be modified as required to connect with the connected application accordingly.

- **NETQRESTJSON**: Contains the default JSON policies for converting XDS to JSON format and vice versa.
- **NETQRESTPWD**: Contains the policies for password synchronization.

Remote Loader

A Remote Loader enables a driver shim to execute outside of the Identity Manager engine (perhaps remotely on a different machine). The Remote Loader is a service that executes the driver shim and passes information between the shim and the Identity Manager engine.

For the REST driver, install the driver shim on the server where the Remote Loader is running. You can choose to use SSL to encrypt the connection between the Identity Manager engine and the Remote Loader. For more information, see “Configuring the Drivers to Run in Remote Mode with SSL” in the *NetIQ Identity Manager Setup Guide for Linux* or “Configuring the Remote Loader and Drivers” in the *NetIQ Identity Manager Setup Guide for Windows*.

Driver Concepts

This section contains the following information:

- “Introduction” on page 15
- “How the Driver Works” on page 18

Introduction

The following concepts are associated with the REST driver:

- “REST” on page 15
- “JSON” on page 16
- “Resource” on page 16
- “Resource Handler” on page 16
- “URL Placeholder” on page 17
- “XML” on page 17
- “HTTP” on page 18
- “HTTPS” on page 18

REST

REST is an HTTP-based protocol for exchanging messages over the network. Since REST is built on HTTP protocol, it supports `POST, PUT, GET, PATCH, DELETE` methods to communicate with the application logic.

JSON

JSON (**Java Script Object Notation**) is a lightweight data-interchange format. JSON stores information in a Key-Value pair format. The Identity Manager driver for REST uses JSON as a data format for payload transfer. For more information about the JSON format used by the driver, see Appendix D, "Supported JSON Format," on page 111.

Resource

A resource is a user, group, or an object that the driver tries to synchronize with the Subscriber and Publisher channels. To be more precise, a REST resource in the driver is a combination of the REST application schema name and the Resource handler. For example, in the URL `http://ipaddress:port/User`, *User* is an example of a REST resource that can be configured to use *Default* as the Resource Handler. For more information, see "Resources" on page 99. To configure a REST resource, Identity Manager provides Driver Configuration options.

Resource Handler

A Resource handler is the mapping of an Identity Manager operation with the REST method. To configure a Resource handler, Identity Manager provides the Driver Configuration options. For more information, see "Resources" on page 99.

A REST call invokes the REST method mapped with an Identity Manager operation. The REST driver supports two Resource handler modes. They are:

- Default - Uses the default HTTP methods for configuring handlers and for managing operations on respective resources. In this mode, the REST driver chooses the best possible mapping for the corresponding Identity Manager operation. For example, an Identity Manager ADD operation corresponds to a POST method and a MODIFY operation corresponds to PUT method of the REST application.

  The REST driver generates the complete URL of a REST method by combining the Base URL for REST Resources and the Schema Name. For example, *https:url.example.com/users*, where *https:url.example.com* is the base URL and *users* is the schema name. Table 1-1 lists the Identity Manager operations, their corresponding default REST methods and the URLs.

Table 1-1 Default Resource Handler

Identity Manager Operation	REST Method	URL
ADD	POST	`http://ipaddress:port/SchemaName<api-version>`
MODIFY	PUT	`http://ipaddress:port/SchemaName/<association><api-version>`
QUERY	GET	`http://ipaddress:port/SchemaName/<association><filter><api-version>`
DELETE	DELETE	`http://ipaddress:port/SchemaName/<association><api-version>`

NOTE: In the GET method, the driver replaces the `<filter>` placeholder by `?search-attr=<searchAttrName1> eq <value1>'` and `<searchAttrName2> eq '<value2'>&read-attr='<readAttr1>`' and '`<readAttr2>`' filter value.

- Custom - Uses the Resource Handler parameters in the Driver Configuration page to customize the driver to suite your deployment scenario. In this mode, the driver generates the complete URL of the REST method by combining the Base URL for REST Resources and the user specific URL in the **URL extension**. For example, *https:url.example.com/users*

URL Placeholder

A URL placeholder is a variable defined in the URL extension within angular brackets. The attribute-value pair in the URL token element of the `driver-operation-data` replaces this placeholder value during the data transfer. For example, consider a sample URL `http://ipaddress:port/SchemaName/<association><api-version>`. During the driver operation, the `<api-version>` URL placeholder is replaced by the value in the element `<url-token api-version="1.0"/>`.

XML

XML (Extensible Markup Language) is a generic subset of Standard Generalized Markup Language (SGML) that allows for exchange of structured data on the Internet.

HTTP

HTTP is a protocol used to request and transmit data over the Internet or other computer network. The protocol works well in an Internet infrastructure and with firewalls.

HTTP is a stateless request/response system because the connection is usually maintained only for the immediate request. The client establishes a TCP connection with the server and sends it a request command. The server then sends back its response.

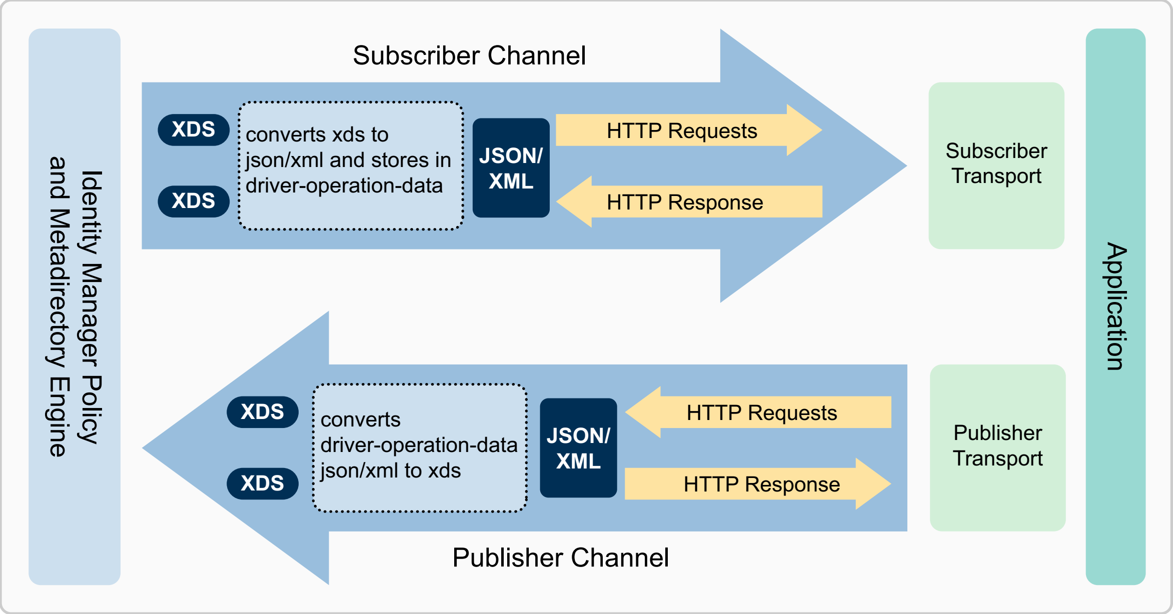
HTTPS

HTTPS is the HTTP protocol over Secure Socket Layer (SSL) as a sub-layer under the regular HTTP application layering. HTTPS encrypts and decrypts user page requests as well as the pages that are returned by the Web server.

How the Driver Works

Figure 1-1 illustrates the data flow between Identity Manager and REST driver:

Figure 1-1 *REST Driver Data Flow*



The Identity Manager engine uses **XDS**, a specialized form of XML, to represent events in the Identity Vault. Identity Manager passes the XDS to the driver policy, which consists of basic policies and DirXML Script.

The driver uses a specialized form of XDS called `<driver-operation-data>`. The `<driver-operation-data>` element encapsulates the metadata and payload for a REST request.

When an event occurs in the Identity Vault, Identity Manager creates an XDS command to represent that event. Identity Manager passes the XDS command to the driver policy. The driver policy transforms that XDS command with an output transformation policy.

This output transformation policy generates the <driver-operation-data> that includes commands, URIs, methods, and payload information for the REST request to successfully complete on the Subscriber channel.

When the request completes, the driver processes responses and reports status of the completed operation to the Identity Manager engine or the Identity Vault.

On the Publisher channel, the REST driver receives the REST request in <driver-operation-data> format. Using the input policy, the driver converts the request to an XDS event and reports back to the connected system.

Understanding Driver Operation Data

The driver shim applies special handling to Subscriber commands based on an XML element embedded in the command, which appears in the driver shim as <driver-operation-data>. The <driver-operation-data> element is added to the command from one of the Subscriber channel policies.

The <driver-operation-data> element includes the metadata with the class-name, command, REST method, and the URI. The command, the REST method, and the URI is required only if the Resource handlers for the resources are not previously configured in the driver parameters. The <request> tag includes the url-token associations, header content-type, and the data to be transferred. The <value> tag includes the JSON payload information.

The <driver-operation-data> element for a REST request includes the following elements:

- **<request>:** Embeds the request information required to make the HTTP call.
- **<url-token>:** Includes the placeholder provided in the driver configuration for resource. For example, during the driver configuration, version is the placeholder added to a resource URL /User/<version>. The attribute-value pair in the URL token element replaces this placeholder. For example, <url-token version="1.0"/>.
- **<header>:** Includes the additional headers that can be added to the REST request in addition to the ones configured in the resource.
- **<value>:** Includes the XML or the JSON payload.

Below is a sample request to add new users with the same common name using the <driver-operation-data> element:

```
<driver-operation-data class-name="User" command="add">
        <request method="put" url="https://172.16.0.0:XXXX/User/rest123">
          <url-token association="rest123"/>
          <header content-type="application/json"/>
          <value>{"CN":[{"value":"rest6789"}],"Full
Name":[{"value":"rest6789 rest6789"}],"Given
Name":[{"value":"rest6789"}],",","Surname":[{"value":"rest6789"}],"Login
Disabled":[{"value":"true"}]}
          </value>
        </request>
        <request method="put" url="https://172.16.0.0:XXXX/User/rest123">
         <url-token association="rest123"/>
          <header content-type="application/json"/>
          <value>{"CN":[{"value":"rest1234"}],"Full
Name":[{"value":"rest1234 rest1234"}],"Given
Name":[{"value":"rest1234"}],",","Surname":[{"value":"rest1234"}],"Login
Disabled":[{"value":"true"}]}
          </value>
        </request>
</driver-operation-data>
```

You will get a response similar to the below sample for this request:

```
<input>
   <driver-operation-data class-name="User" command="add" remote-
host="172.16.0.0" url="http://172.16.0.0:XXXX/User">
        <header content-type="application/json"/>
      <response>
        <value>{"association":"noble2","CN":"noble2","Full
Name":"noble2","Given
Name":"noble2","nspmDistributionPassword":"novell@123","Surname":"noble2"}
        </value>
      </response>
    </driver-operation-data>
</input>
```

NOTE: The driver retains the <`driver-operation-data`> between any REST operations. The connected application appends its response to the same <`driver-operation-data`> and returns it back to the driver shim. A single <`driver-operation-data`> element is capable of accommodating multiple requests that belong to the same class.

Response Headers

When a REST call is made to a REST service, a response is returned with tokens appended to the header tag elements. These type of responses with tokens appended in the header tag are called response headers.

The `response-header` tag in a driver trace is shown in the following example:

```
<nds dtdversion="3.0">
  <source>
    <product build="XXXXXXX" version="1.1.2.0">Identity Manager REST
Driver</product>
    <contact>NetIQ Corporation.</contact>
  </source>
  <output>
    <status event-id="sles12sp2-name1-130177#20201015102623#1#2:45f7d829-
f562-4745-aa97-29d8f74562f5" level="success" type="driver-general">
      <driver-operation-data class-name="User" command="add" dest-
dn="\SLES12SP2_USRNAME_130177_TREE\data\users\netiq26" event-
id="sles12sp2-usrname-130177#20201015102623#1#2:45f7d829-f562-4745-aa97-
29d8f74562f5" src-dn="\SLES12SP2_USRNAME_130177_TREE\data\users\netiq26">
        <response>
          <url-token/>
          <content-type="application/json"/>
          <response-header Cache-Control="no-cache,must-revalidate,max-
age=0,no-store,private" Content-Type="application/json;charset=UTF-8"
Date="Thu, 15 Oct 2020 10:26:38 GMT" Expect-CT='max-age=86400, report-
uri="https://a.forcesslreports.com/Expect-CT-report/nullm"/>
          <value message="Created"
status="201">{"id":"0052v00000hxEXOAA2","success":true,"errors":[]}</
value>
        </response>
      </driver-operation-data>
      <operation-data association="" src-
dn="\SLES12SP2_USRNAME_130177_TREE\data\users\netiq26">
        <password-subscribe-status>
          <association/>
        </password-subscribe-status>
      </operation-data>
    </status>
  </output>
</nds>
```

In the above example, the tokens that are appended to the `response-header` tag are, `Cache-Control`, `Content-Type`, `Date`, `Expect-CT` and `report-uri`.

Support for Standard Driver Features

The following sections provide information about how the REST driver supports the standard driver features:

- "Supported Operations" on page 22
- "Local Platforms" on page 22
- "Remote Platforms" on page 22
- "Supporting Driver Authentication" on page 22
- "Supporting Publish Mode" on page 23
- "Supporting Identity Manager Engine as a REST EndPoint" on page 24
- "Synchronizing Information" on page 25

Supported Operations

The REST driver performs the following operations on the Publisher and Subscriber channels:

- **Publisher Channel:** Add, Modify, Delete, and Query operations on User and Group objects, and password synchronization.
- **Subscriber Channel:** Add, Modify, Delete, Migrate, and Query operations on User and Group objects, Password Set/Reset operations only on User objects.

Local Platforms

A local installation is an installation of the driver on the Identity Manager server. The REST driver can be installed on the operating systems supported for the Identity Manager server.

For information about the operating systems supported for the Identity Manager server, see the NetIQ Identity Manager Technical Information website (https://www.netiq.com/products/identity-manager/advanced/technical-information/).

Remote Platforms

The REST driver can use the Remote Loader service to run on a server other than the Identity Manager server. The REST driver can be installed on the operating systems supported for the Remote Loader.

For information about the supported operating systems, see the NetIQ Identity Manager Technical Information website (https://www.netiq.com/products/identity-manager/advanced/technical-information/).

Supporting Driver Authentication

The REST driver allows you to configure the following authentication methods. By default the REST driver supports Basic authentication method. However, you can change the authentication method using the Driver configuration.

- **Anonymous:** The driver uses anonymous authentication method for authenticating to a RESTful service. On the Subscriber channel, this method allows valid connectivity between the REST driver and any RESTful service that supports anonymous authentication method. On the Publisher channel, the driver allows anonymous access to the Identity Vault for any RESTful service.
- **Basic:** The driver uses the ID and password that you specify during driver configuration for authenticating to the RESTful service. The driver considers the Publisher user credentials as the basic authentication method credentials. In this authentication method, the driver uses these credentials to connect to the endpoints exposed on the Publisher channel.
- **OAuth2.0:** The OAuth 2.0 is an open authentication protocol that enables any third-party application to access data from an HTTP service to share data among various applications. The driver supports OAuth2.0 authentication only on the Subscriber channel.

Secured communication between client-server applications is established using authorization tokens such as, JSON Web Tokens, Access Tokens and Refresh Tokens.

- **JSON Web Token (JWT)**: JWT defines a compact and self-contained way for securely transmitting information between parties. JWTs can be encrypted to provide secrecy between client-server applications. For more information on JWT see, RFC7519.
- **Access Token**: Access tokens carry the necessary information to access a resource directly.
- **Refresh Token**: Refresh tokens carry the necessary information to get a new access token. If an access token is expired, refresh token allows the application to obtain a new access token without user's intervention. Refresh tokens have the potential for a longer lifetime, whereas access tokens have a comparatively shorter lifetime.

IMPORTANT: For any operation performed on the connected application using OAuth 2.0, an access token is sent for authorization of the user from the connected application. The access token expires post the session idle time set for the connected application, or in case of a system restart. The session idle time is configurable as per requirement. The connected application displays Unauthozied Access error or an Invalid Session error for any request initiated with an expired access token.The presence of a refresh token helps to re-establish the failed session internally, by generating a new access token without the user having to log in again.

The resource owner grants authorization to a client application in cooperation with the authorization server associated with the resource server. The resource owner grants authorization to a client application using a in cooperation with the authorization server associated with the resource server. When requesting for authorization, the client receives an authorization grant from the resource owner. An authorization grant is an authorization credential representing the resource owner authorization in the form of a JSON Web Token (JWT). The two authorization grants supported by the REST driver are resource owner password credentials and client credentials.

- Client Credentials - Uses the client ID and secret received while registering with the identity provider.
- Resource Owner Password - Shares the resource owner credentials with the client application. Uses the user name and password of the resource owner as authorization grant to obtain an access token. For example, you can use your Twitter user name and password to log in to a client application.

NOTE: Ensure that you set the appropriate query options while configuring the authorization query in the driver parameters. For more information, see “Subscriber Settings” on page 95.

Supporting Publish Mode

The Identity Manager driver for REST supports Publish as Publisher option.

If **Publish** is selected, the driver exposes the REST endpoints to receive the events from the connected RESTful service and then pushes the events to the Identity Vault.

Supporting Identity Manager Engine as a REST EndPoint

The REST driver exposes REST endpoints to the Identity Manager engine. This facilitates easy communication between external applications and services with eDirectory and Identity Manager engine via the REST API.

NOTE: The authentication header and content type are mandatory for REST methods.

Table 1-2 lists an example of POST REST method that the driver supports for a User class:

Table 1-2 *POST Method*

METHOD: POST	
User URI	http://ipaddress:port/User
Payload	{"association":"User2","Postal Code":["324324324"],"Surname":["User2"],"CN":["User2"]}
Authorization	Basic c3lzdGVtL3N5c3RlbQ==
Content-Type	application/json
Response	201 Created

Table 1-3 lists an example of DELETE REST method that the driver supports:

Table 1-3 *DELETE Method*

METHOD: DELETE	
User URI	http://ipaddress:port/User/User2
Payload	Not required
Authorization	Basic c3lzdGVtL3N5c3RlbQ==
Content-Type	application/json
Response	200 OK

Table 1-4 lists an example of PUT REST method that the driver supports:

Table 1-4 *PUT Method*

METHOD: PUT	
User URI	http://ipaddress:portUser/User2
Authorization	Basic c3lzdGVtL3N5c3RlbQ==
Content-Type	application/json
Payload	{"Title":[{"add":["Manager"]}]}
Response	204 No Content

Table 1-5 lists an example of GET REST method that the driver supports:

Table 1-5 *GET Method*

METHOD: GET	
User URI	http://ipaddress:port/User?search-attr=given name eq ‘test*user’ and cn eq ‘test*’&read-attr=title
Payload	Not Applicable
Authorization	Basic c3lzdGVtL3N5c3RlbQ==
Content-Type	application/json
Response	{ "totalResults": 1, "results": [{ "src-dn": "\\GEN-REST1\\system\\servers\\TestUser", "class-name": "User", "Title": ["SE"] }]}

Synchronizing Information

Unlike most other drivers, the REST driver synchronizes protocols instead of objects. The driver includes the following features:

- HTTP transport of data between the Identity Vault and a Web service
- SSL connections using the HTTPS protocol
- Subscriber HTTP and HTTPS proxy servers
- Potential to act as an HTTP or HTTPS listener for incoming connections on the Publisher channel
- Potential extensibility through customized Java code

  For more information, see Appendix B, “Using Java Extensions,” on page 105.

2 Installing the Driver Files

You can install the REST driver on the Identity Manager server or on a remote server using the Remote Loader.

To install the driver, you first need to install the driver files, install the driver packages, and then modify the driver configuration to suit your environment. This section describes how to install the driver files. For information on installing and configuring driver packages, see Chapter 3, “Creating A New Driver Object,” on page 29.

- “Prerequisites for Driver Installation” on page 27
- “Installing the REST Driver Files” on page 27

Prerequisites for Driver Installation

The installation and configuration process for the driver requires Identity Manager and/or Remote Loader, and Designer for Identity Manager. Before installing the driver, ensure that you download the following software to your Identity Manager environment:

- Identity Manager 4.8 or later
- Designer 4.8 or later

Installing the REST Driver Files

You can install the REST driver files as a root user or as a non-root user in your system. The procedure to install the driver files is similar for any connected application.

You must ensure that you have the required REST driver files such as,**`.zip, .rpm, and.jar`** etc., from the required driver build available in Micro Focus Download site to install the REST driver in your system.

For example:

- `.zip` file: <*`IDM_REST_1100.zip`*>
- `.rpm` file: <*`netiq-DXMLRESTDrv.rpm`*>
- `.jar` file: <*`RESTUtils.jar`*>

This section explains the common procedure to install the driver files:

1. Download and unzip the contents of the ***`<IDM_REST_1100.zip>`*** file to a temporary location on your computer.
2. Install the driver files (for IDM 4.7.4 and above) based on your user role.

   To install as a:

   - root user, see “Installing Driver Files as a Root User” on page 28.
   - non-root user, see “Installing Driver Files as a Non-Root User” on page 28.

Installing Driver Files as a Root User

1. Login as a root user on the server where you want apply the driver jar file.
2. Navigate to the extracted ***<IDM_REST_1100.zip>*** directory and perform one of the following actions based on your platform:
   - **Linux**: Install the new ***<netiq-DXMLRESTDrv.rpm>*** in your driver installation directory by running the following command in a terminal window:

     <rpm -Ivh (binaries-path)/netiq-DXMLRESTDrv.rpm>
   - **Windows**: Copy the *<RESTCommon.jar>*, *<RESTDriverShim.jar>*, *<RESTUtil.jar>* files to your driver installation folder. For example, `\NetIQ\IdentityManager\NDS` (local installation) or `\Novell\RemoteLoader\64bit` (remote installation).

Installing Driver Files as a Non-Root User

1. Verify that the **/rpm** directory exists and contains the **_db.000** file.
2. The **_db.000** file is created during a non-root installation of the Identity Manager engine. The absence of this file indicates that the Identity Manager is not installed properly. In such case, reinstall the Identity Manager to correctly place the file in the mentioned directory.
3. To set the root directory to the location of non-root in Identity Manager, enter the following command in the command prompt:

   ROOTDIR=<non-root eDirectory location>

   This will set the environmental variables to the directory where Identity Manager is installed as a non-root user.
4. For example, to install the REST driver rpm, use this command:

   rpm --dbpath $ROOTDIR/rpm -Ivh --relocate=/usr=$ROOTDIR/opt/novell/eDirectory --relocate=/etc=$ROOTDIR/etc --relocate=/opt/novell/eDirectory=$ROOTDIR/opt/novell/eDirectory --relocate=/opt/novell/dirxml=$ROOTDIR/opt/novell/dirxml --relocate=/var=$ROOTDIR/var --badreloc --nodeps --replacefiles /home/user/netiq-DXMLRESTDrv.rpm

   NOTE: In the above command **/opt/novell/eDirectory** is the location where non-root Identity Manager is installed, and **/home/user/** is the home directory of the non-root user.

3 (Conditional) If the driver is running locally, start the Identity Manager and the driver instance.

4 (Conditional) If the driver is running with a Remote Loader instance, start the Remote Loader instance and the driver instance.

You can also install the REST driver files on the Identity Manager server or a remote server that supports Remote Loader configuration. For more information about installing Remote Loader, see “Considerations for Installing Identity Manager Engine Components and Remote Loader” in the *NetIQ Identity Manager Setup Guide for Linux* or “Planning to Install the Remote Loader” in the *NetIQ Identity Manager Setup Guide for Windows*.

3 Creating A New Driver Object

After the REST driver files are installed on the server where you want to run the driver (see Chapter 2, “Installing the Driver Files,” on page 27), you can create the driver in Designer. You do so by installing the driver packages and then modifying the driver configuration to suit your environment.

The following sections provide instructions to create the driver:

- “Creating the Driver Object in Designer” on page 29
- “Activating the Driver” on page 44
- “Adding Packages to an Existing Driver” on page 44

Creating the Driver Object in Designer

The Designer tool helps you to create the REST driver object. You need to install the driver packages and then modify the configuration to suit your environment. After you create and configure the driver, you need to deploy it to the Identity Vault and start it.

- “Importing the Current Driver Packages” on page 29
- “Installing the Driver Packages” on page 31
- “Configuring the Driver Object” on page 41
- “Deploying the Driver Object” on page 42
- “Starting the Driver” on page 43

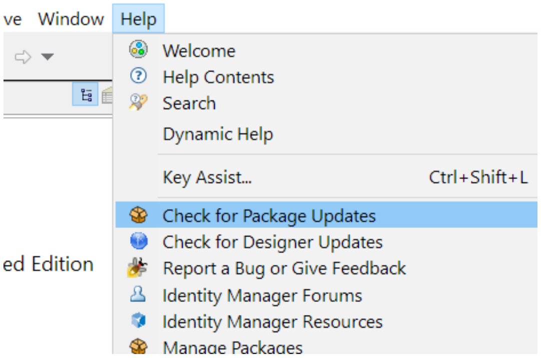
NOTE: NetIQ recommends that you use the new package management features provided in Designer to create the REST driver. You should not create the driver objects by using the new Identity Manager 4.0 and later or configuration files through Identity Console. This method of creating driver objects is no longer supported.

Importing the Current Driver Packages

The driver packages contain the items required to create a driver, such as policies, entitlements, filters, and Schema Mapping policies. These packages are only available in Designer and can be updated after they are initially installed. You must have the most current version of the packages in the Package Catalog before you can create a new driver object.

To verify that you have the most recent version of the driver packages in the Package Catalog:

1. Open Designer.
2. In the toolbar, click **Help** > **Check for Package Updates**.


ve Window Help
Welcome
Help Contents
Search
Dynamic Help
Key Assist... Ctrl+Shift+L
Check for Package Updates
Check for Designer Updates
Report a Bug or Give Feedback
Identity Manager Forums
Identity Manager Resources
Manage Packages
ed Edition

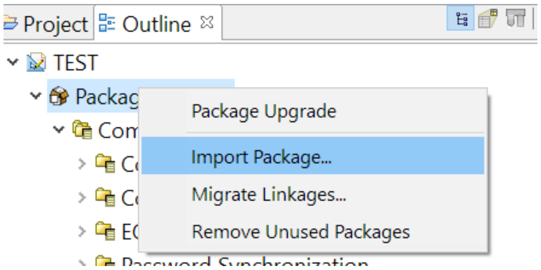
3 Click **OK** to update the packages

or

Click **OK** if the packages are up-to-date.

4 In the Outline view, right-click the **Package Catalog**.

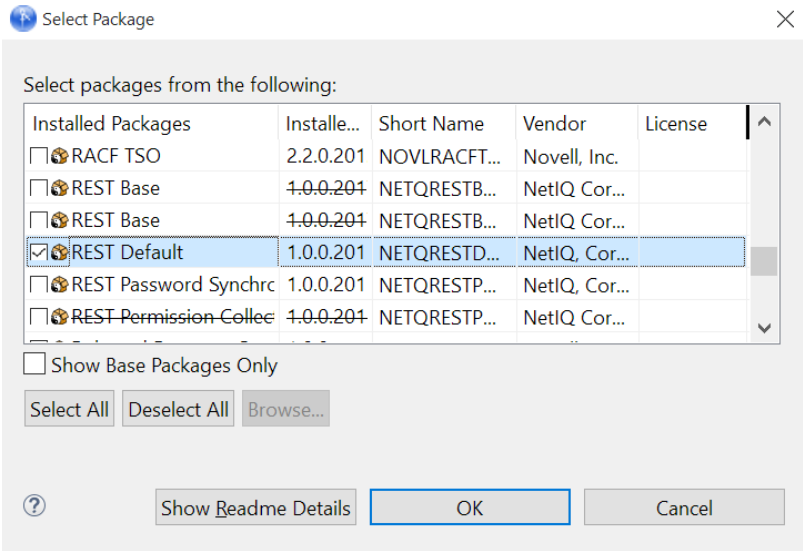
5 Click **Import Package**.



6 Select any REST driver package

or

Click **Select All** to import all of the packages displayed.



Select packages from the following:

Installed Packages	Installe...	Short Name	Vendor	License
☐ RACF TSO	2.2.0.201	NOVLRACFT...	Novell, Inc.	
☐ REST Base	~~1.0.0.201~~	NETQRESTB...	NetIQ Cor...	
☐ REST Base	~~1.0.0.201~~	NETQRESTB...	NetIQ Cor...	
☑ REST Default	1.0.0.201	NETQRESTD...	NetIQ, Cor...	
☐ REST Password Synchrc	1.0.0.201	NETQRESTP...	NetIQ, Cor...	
☐ ~~REST Permission Collec~~	~~1.0.0.201~~	NETQRESTP...	NetIQ Cor...	

☐ Show Base Packages Only

By default, only the base packages are displayed. Deselect **Show Base Packages Only** to display all packages.

7 Click **OK** to import the selected packages, then click **OK** in the successfully imported packages message.

8 After the current packages are imported, continue with "Installing the Driver Packages" on page 31.

Installing the Driver Packages

After you have imported the current driver packages into the Package Catalog, you can install the driver packages to create a new driver, or update the existing driver package. To install driver packages, you have to set-up Identity Vault and the driver set.

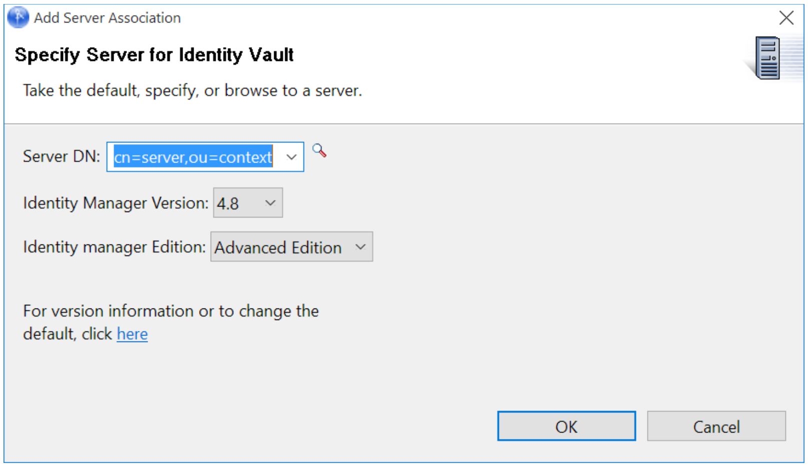
Setting up Identity Vault

1 In **Designer > Outline** view, open your project.

2 Right click project > **New** > **Identity Vault**, or drag and drop Identity Vault from the Palette to Modeler window.

The Add Server Association screen appears.

3 In the Add Server Association screen, select the following field values and click **OK**.



Add Server Association

Specify Server for Identity Vault

Take the default, specify, or browse to a server.

Server DN: cn=server,ou=context

Identity Manager Version: 4.8

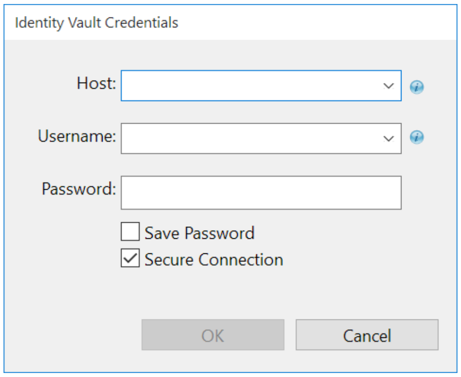
Identity manager Edition: Advanced Edition

For version information or to change the default, click here

- Server DN
- Identity Manager Version
- Identity Manager Edition

The Identity Vault Credentials window appears.

4 In Identity Vault Credentials window, enter values as shown in the following table.



Identity Vault Credentials

Host:

Username:

Password:

☐ Save Password

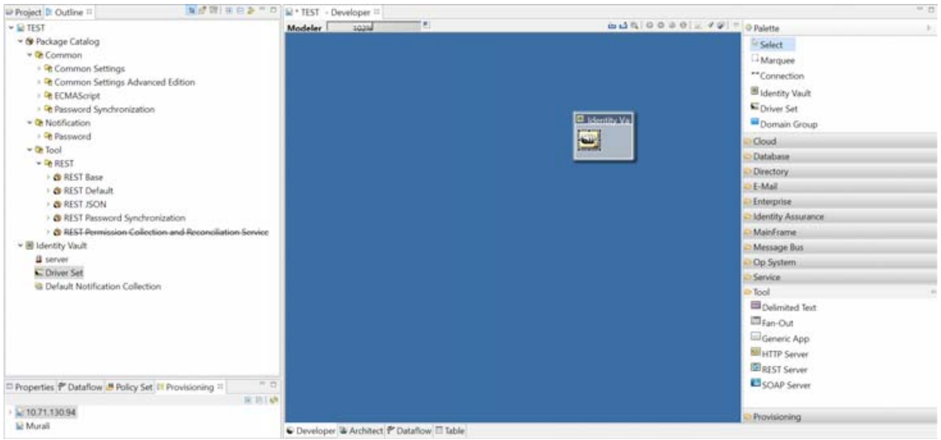
☑ Secure Connection

Field	Description
Host	The identity vault hosting machine's IP address.
Username	The name of the user, for example, Admin, if the user is an administrator.
Password	Password of user to login to the identity vault.

5 Select **Save Password**, if you want to save your password for easy logins in the future.

6 Click **OK**.

The Identity Vault and the Driver Set appears in the Modeler window as shown in the following image.



7 In the right pane, drag and drop the REST Server from the **Tools** tab to the Modeler.

8 In the Driver Configuration Wizard, select **REST Base** (Contains the base functionality for a driver. You must install a driver base configuration package first).

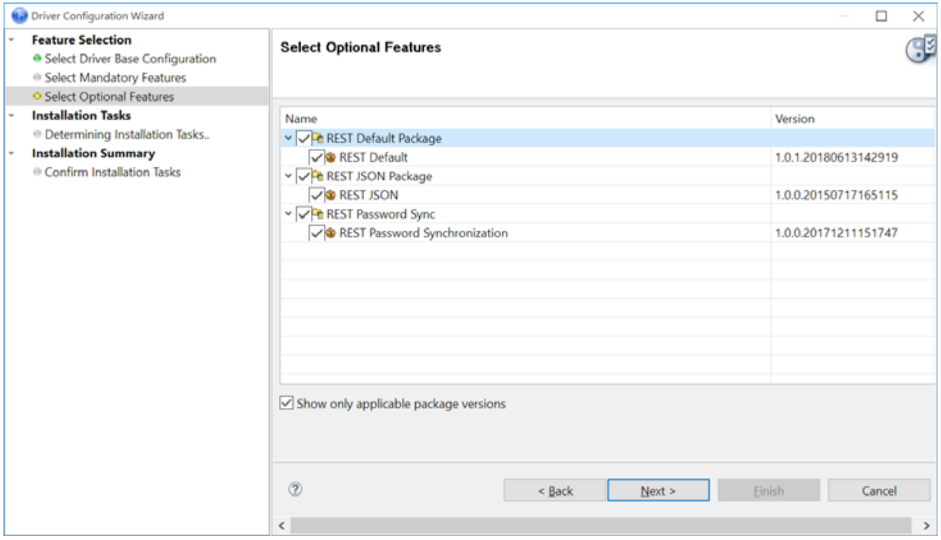
NOTE: You can only select one base package.

9 Click **Next**.

10 Select the optional features to install for the REST driver, the options are:

- REST Default Package
- REST JSON Package: This package contains the default JSON configurations

- REST Password Sync: This packages contains the policies that enable the REST driver to synchronize passwords. If you want to synchronize passwords, verify that this option is selected. For more information, see the *NetIQ Identity Manager Password Management Guide*.


Driver Configuration Wizard
Feature Selection
Select Driver Base Configuration
Select Mandatory Features
Select Optional Features
Installation Tasks
Determining Installation Tasks..
Installation Summary
Confirm Installation Tasks
< Back
Next >
Finish
Cancel

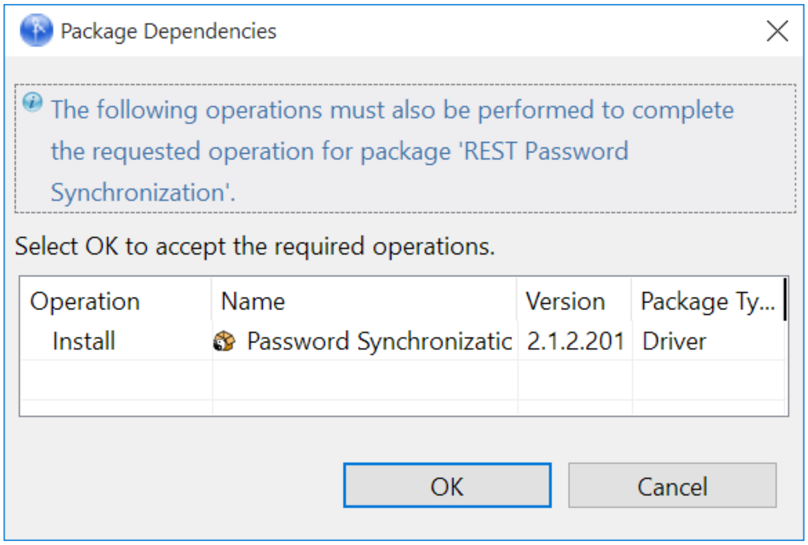
Select Optional Features

Name	Version
☑ REST Default Package	
☑ REST Default	1.0.1.20180613142919
☑ REST JSON Package	
☑ REST JSON	1.0.0.20150717165115
☑ REST Password Sync	
☑ REST Password Synchronization	1.0.0.20171211151747

☑ Show only applicable package versions

11 Click **Next**.

The package dependencies window appears.

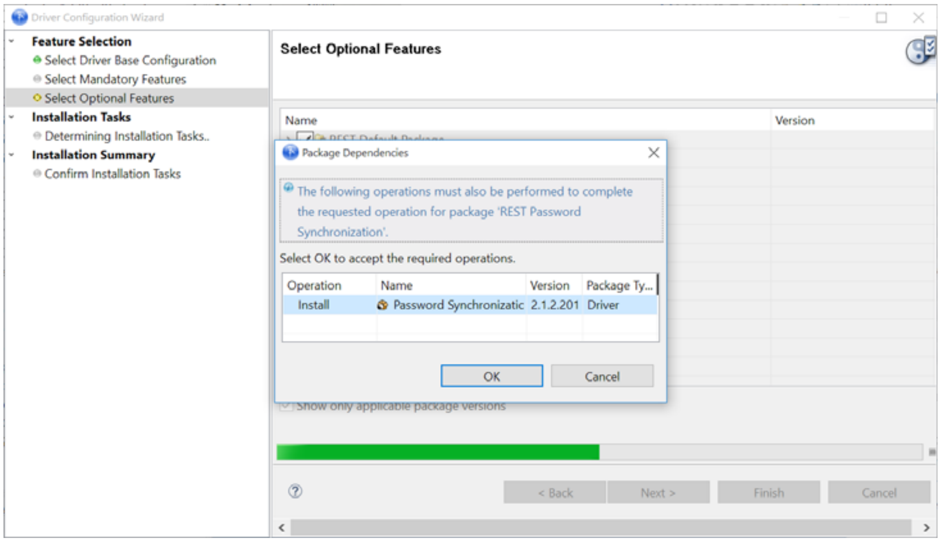


The following operations must also be performed to complete the requested operation for package 'REST Password Synchronization'.

Select OK to accept the required operations.

Operation	Name	Version	Package Ty...
Install	Password Synchronizatic	2.1.2.201	Driver

12 (Conditional) Click **OK** to install the package dependency listed.


Driver Configuration Wizard
Feature Selection
Select Driver Base Configuration
Select Mandatory Features
Select Optional Features
Installation Tasks
Determining Installation Tasks..
Installation Summary
Confirm Installation Tasks
Select Optional Features
Name
Version
Package Dependencies
OK
Cancel
Show only applicable package versions
< Back
Next >
Finish
Cancel

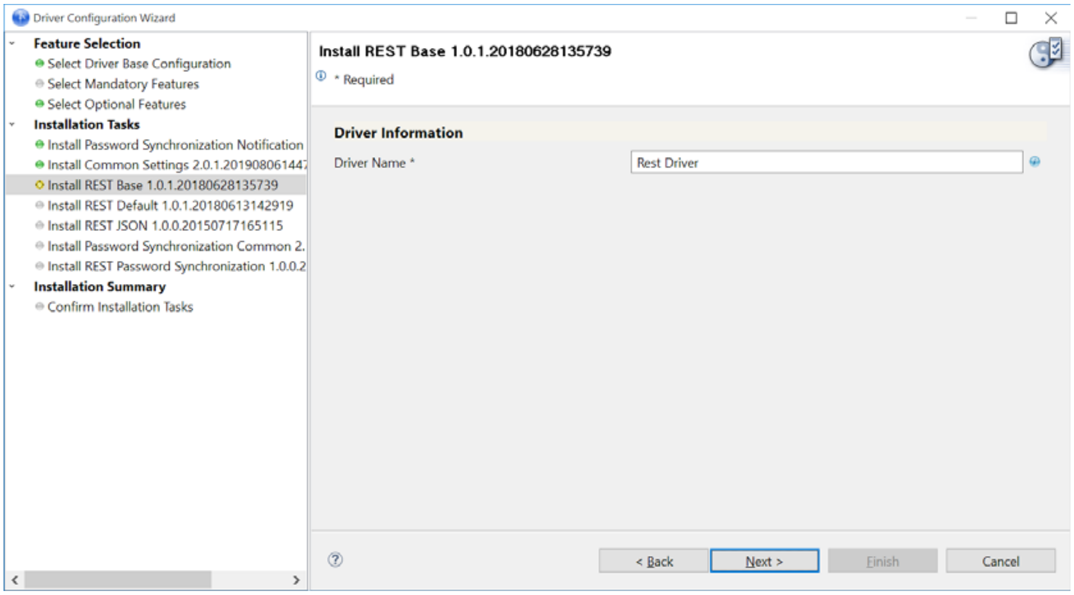
The following operations must also be performed to complete the requested operation for package 'REST Password Synchronization'.

Select OK to accept the required operations.

Operation	Name	Version	Package Ty...
Install	Password Synchronizatic	2.1.2.201	Driver

NOTE: If there are any dependent packages associated with the selected package, you must install them to proceed.

13 On the Driver Information page, specify a name for the driver, then click **Next**.


Driver Configuration Wizard
Feature Selection
Select Driver Base Configuration
Select Mandatory Features
Select Optional Features
Installation Tasks
Install Password Synchronization Notification
Install Common Settings 2.0.1.20190806144
Install REST Base 1.0.1.20180628135739
Install REST Default 1.0.1.20180613142919
Install REST JSON 1.0.0.20150717165115
Install Password Synchronization Common 2.
Install REST Password Synchronization 1.0.0.2
Installation Summary
Confirm Installation Tasks
< Back
Next >
Finish
Cancel

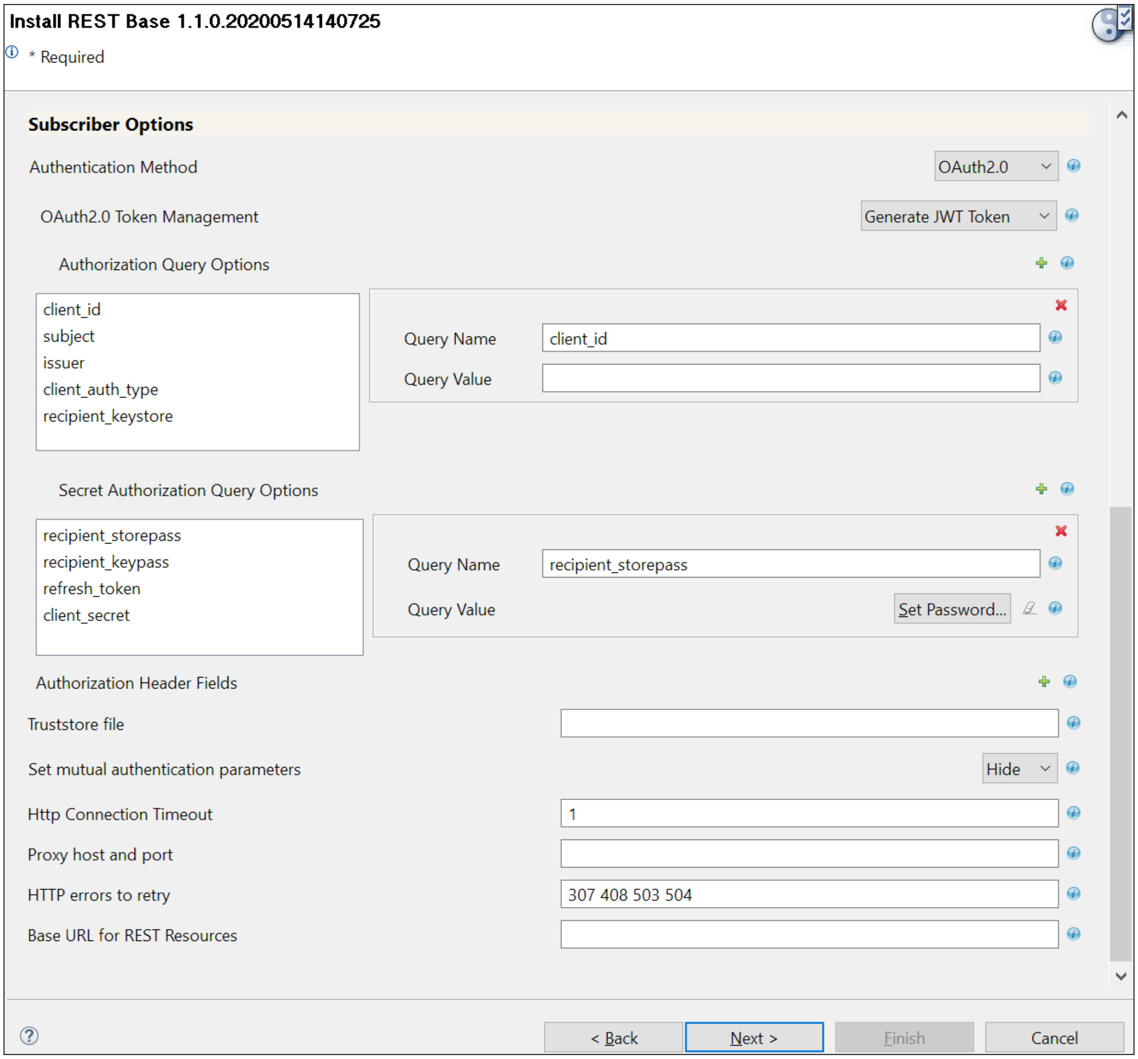
Install REST Base 1.0.1.20180628135739

* Required

Driver Information

Driver Name * Rest Driver

14 On the Install REST Base page, fill in the following fields for the Subscriber options, and click **Next**.



Install REST Base 1.1.0.20200514140725

* Required

Subscriber Options

Authentication Method: OAuth2.0

OAuth2.0 Token Management: Generate JWT Token

Authorization Query Options

client_id
subject
issuer
client_auth_type
recipient_keystore

Query Name: client_id

Query Value:

Secret Authorization Query Options

recipient_storepass
recipient_keypass
refresh_token
client_secret

Query Name: recipient_storepass

Query Value: Set Password...

Authorization Header Fields

Truststore file:

Set mutual authentication parameters: Hide

Http Connection Timeout: 1

Proxy host and port:

HTTP errors to retry: 307 408 503 504

Base URL for REST Resources:

The Authentication Methods available are, **Anonymous**, **Basic**, and **OAuth2.0**, and the screen defaults to **Basic**. Based on the selection you make the other fields appear.

NOTE: Fields marked with ** indicate common fields that appear for all Authentication Methods.

- If you select **Basic**, the following fields appear:
  - **Authentication ID**: Specify the authentication ID for Basic Authorization (on the HTTP header) is used.
  - **Authentication Password**: Specify the authentication password for Basic Authorization (on the HTTP header) is used.
  - **Authorization Header Fields****: Click the + icon to create authentication header fields. Enter the required authentication header fields and supported values for the selected authentication method.
  - **Truststore file****: Specify the path and the name of the keystore file that contains the trusted certificates for the remote server to provide server authentication. For example, `C:\security\truststore`. Leave this field blank when server authentication is not used.

    - Set mutual authentication parameters**: Select Show if you want to set mutual authentication information.
        - Keystore file: Specify the path and the name of the keystore file that contains the trusted certificates for the remote server to provide mutual authentication. For example, `C:\security\keystore`. Leave this field blank when mutual authentication is not used.
        - Keystore password: Specify the password for the keystore file. Leave this field blank when mutual authentication is not used.
    - Http Connection Timeout**: Specify the HTTP connection time out value. The driver waits for the time specified and terminates the HTTP connection. The timeout value must be greater than zero.
    - Proxy host and port**: Specify the host address and the host post when a proxy host and port are used. For example: 192.168.0.0:port. Choose an unused port number on your server. Otherwise, leave this field blank.
    - HTTP errors to retry**: Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces. For example, 307 408 503 504.
    - Base URL for REST**: Specify the URL of the REST server or Web service.
- If you select OAuth2.0, the following fields appear:
    - **OAuth 2.0 Token Management**: Select the token type as required. The available options are, Generate Bearer Token, Generate JWT Token, Enter Bearer Token.
        - Generate Bearer Token: It is an access token issued by servers to achieve multi-server authentication.

          If you select Generate Bearer Token, the following fields appear:

Field	Description
Access Token URL	Specify the URL of the server used for requesting token access.
User Name	*<username to login to the connected application>*
User Password	*<password to login to connected application>*
Authorization Query Options	◆ **`grant_type`**: It is the method the application procures an access token. Enter the value as `password`. ◆ **`client_id`**: The `client_id` is a public identifier for the connected application. Enter the *<client identification value>*. For example: *<3MVG97quAmFZJfVwk3ylU.8elhRYBqG9h25m3TWewozjKnFIY0HrhOEJl7LMET9HHocaHnTB1k04kophr1CgW>* ◆ **`issuer`**: The authorization server's URL that uses the https protocol.

Field	Description
Secret Authorization Query Options **NOTE:** The * indicates mandatory fields and ^ indicates non mandatory fields.	These parameters are set to configure a refresh token. Though not mandatory, if configured the set refresh token value is not overridden with the new value when the access token expires. This may cause login issues until the new refresh token is added. - **`refresh_token^`**: Refresh Token is a web token to acquire new access tokens when current access tokens expire or become invalid. The authorization server of the connected system provides refresh tokens to the Identity Manager to obtain new access token, without user interaction in the backend. - **`client_secret^`**: It is a secret pass phrase associated with the refresh token.

- Generate JWT Token: The JSON Web token is an access request token in the JSON Web Token (JWT) format. It is an encrypted data string consisting of a header, payload, and a signature, and is used to transfer authorization data in client-server applications to authenticate the resource identity.

  If you select Generate JWT Token, the following fields appear:

Field	Description
Authorization Query Options	- **`client_id`** - **`subject`**: The user's unique identity for which the access token is being requested. - **`issuer`** - **`client_auth_type`**: The client's authorization types configured for granting access to the application. - **`recipient_keystore`**: The keystore recipient alias used to look up the digital signature which contains the public key in connected application.
Secret Authorization Query Options	- **`recipient_storepass`**: Password for the recipient keystore. - **`recipient_keypass`**: Password for the recipient key value. - **`refresh_token^`** - **`client_secret^`**

- Enter Bearer Token: Enter a bearer token if you already have one, and configure the `refresh_token` and `client_secret` and set the passwords accordingly.

If you select **Enter Bearer Token**, the following fields appear:

Field	Description
Bearer Token ID	Enter the available bearer token.
Authorization Query Options	- `client_id` - `issuer`
Secret Authorization Query Options	- `refresh_token*`: It is mandatory to configure for an available bearer token. - `client_secret*`

- If you select **Anonymous**: only **Authorization Header Fields**, **Truststore file**, **Set mutual authentication parameters**, **Http Connection Timeout**, **Proxy host and port**, **HTTP errors to retry**, and **Base URL for REST Resources** fields appear.

15 On the Install REST Base page, for the Publisher Options fill in the following fields, then click **Next**.

Field	Description
Publisher Setting	Specify the publisher setting for the REST driver. Based on the selection the other fields appear. The available options are: - **Poll** - **Publish**

Field	Description
These fields appear if **Publisher** is selected.	***Publisher Options***: - **Listening IP address and port**: Specify the IP address of the server where this driver is installed and the port that this driver listens on. You can specify 127.0.0.1, if there is only one network card installed in the server. Choose an unused port number on your server. For example: 127.0.0.1:port. The driver listens on this address for incoming requests, processes the requests, and returns a result. - **Authentication Method, Authentication ID and Authentication Password**: Select the authentication values respectively for the REST driver. The authentication methods available are Anonymous and Basic. You need to specify additional parameters depending upon the selected authentication method. For more information, see "Driver Configuration" on page 93. ***Other options:*** - **KMO name**: When this server is configured to accept HTTPS connections, this is the KMO name in eDirectory. The KMO name is the name before the - in the RDN. Leave this field blank when a keystore file is issued or when HTTPS connections are not used. - **Keystore file**: When this server is configured to accept HTTPS connections, this is the path and the name of the keystore file. For example; `C:\security\keystore`. Leave this field blank when a KMO name is used or when HTTPS connections are not used. - **Keystore password**: When this server is configured to accept HTTPS connections, this is the keystore file password. Leave this field blank when a KMO name is used or when HTTPS connections are not used. - **Server key alias**: When this server is configured to accept HTTPS connections, this is the key alias. Leave this field blank when a KMO name is used or when HTTPS connections are not used. - **Server key password**: When this server is configured to accept HTTPS connections, this is the key alias password (not the keystore password). Leave this field blank when a KMO name is used or when HTTPS connections are not used. - **Require Mutual authentication**: When using SSL, it is common to do only server authentication. However, if you want to force both client and server to present certificates during the handshake process, select Required. - **Heartbeat interval in minutes**: Heartbeat is the interval to be specified for data synchronization between Identity Manager and the connected system. Leave this field blank to turn off the heartbeat.

Field	Description
These fields appear if **Poll** is selected.	***Configure Resource for poll:*** ◆ **Schema name**: Specify the class name of the user resource returned present in application schema. ◆ **Service Endpoints**: Specify the REST URLs for the resource. Mention queryable strings as %s. ◆ **Method**: Select the HTTP method to be used. ◆ **Polling interval in minutes**: Specify the polling interval in minutes. Default is one minute.

16 (Conditional) Fill in the following fields for the Remote Loader information, then click **Next**.

To Connect To Remote Loader:

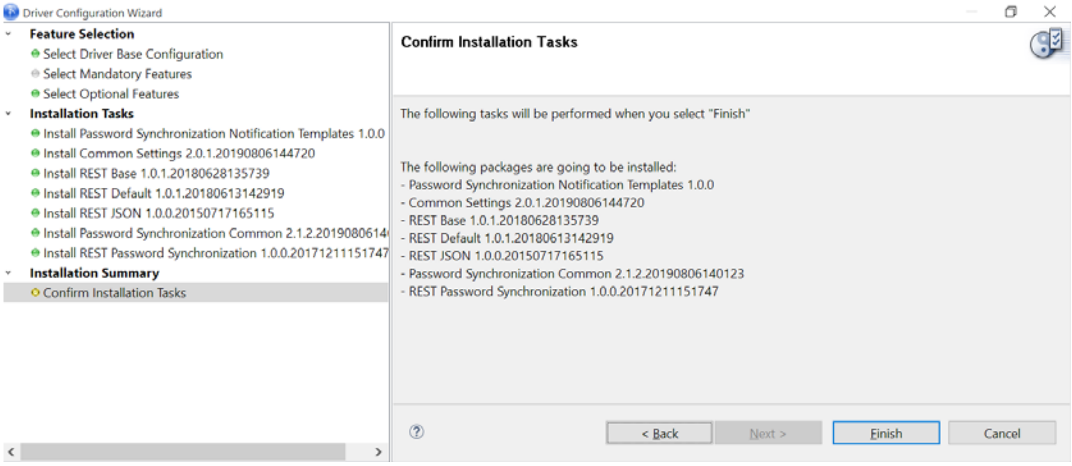
16a Select **Yes** or **No** to determine if the driver will use the Remote Loader.

16b If you select **No**, skip to Step 12.

16c If you select **Yes**, use the following information to complete the configuration of the Remote Loader:

Field	Description
Host Name	Specify the IP address or DNS name of the server where the Remote Loader is installed and running.
Port	Specify the port number for this driver. Each driver connects to the Remote Loader on a separate port. The default value is 8090.
KMO	Specify the key name of the Key Material Object that includes keys and certificates for SSL. You use this parameter only when an SSL connection exists between the Remote Loader and the Identity Manager engine. **NOTE:** When this server is configured to accept HTTPS connections, this is the KMO name in eDirectory. The KMO name is the name before the - in the RDN. Leave this field blank when a keystore file is issued or when HTTPS connections are not used.
Other Parameters	Specify any other parameter required in the connection string. The parameter must be a key-value pair. For example, paraName1=paraValue1
Remote Loader Password	Specify a password to control access to the Remote Loader. It must be the same password that is specified as the Remote Loader password on the Remote Loader.
Driver Password	Specify a password for the driver to authenticate to the Identity Manager server. It must be the same password that is specified as the Driver Object Password on the Remote Loader.

17 Review the summary of tasks that will be completed to create the driver, then click **Finish**.


Driver Configuration Wizard
Feature Selection
Select Driver Base Configuration
Select Mandatory Features
Select Optional Features
Installation Tasks
Install Password Synchronization Notification Templates 1.0.0
Install Common Settings 2.0.1.20190806144720
Install REST Base 1.0.1.20180628135739
Install REST Default 1.0.1.20180613142919
Install REST JSON 1.0.0.20150717165115
Install Password Synchronization Common 2.1.2.2019080614
Install REST Password Synchronization 1.0.0.20171211151747
Installation Summary
Confirm Installation Tasks
< Back
Next >
Finish
Cancel

Confirm Installation Tasks

The following tasks will be performed when you select "Finish"

The following packages are going to be installed:
- Password Synchronization Notification Templates 1.0.0
- Common Settings 2.0.1.20190806144720
- REST Base 1.0.1.20180628135739
- REST Default 1.0.1.20180613142919
- REST JSON 1.0.0.20150717165115
- Password Synchronization Common 2.1.2.20190806140123
- REST Password Synchronization 1.0.0.20171211151747

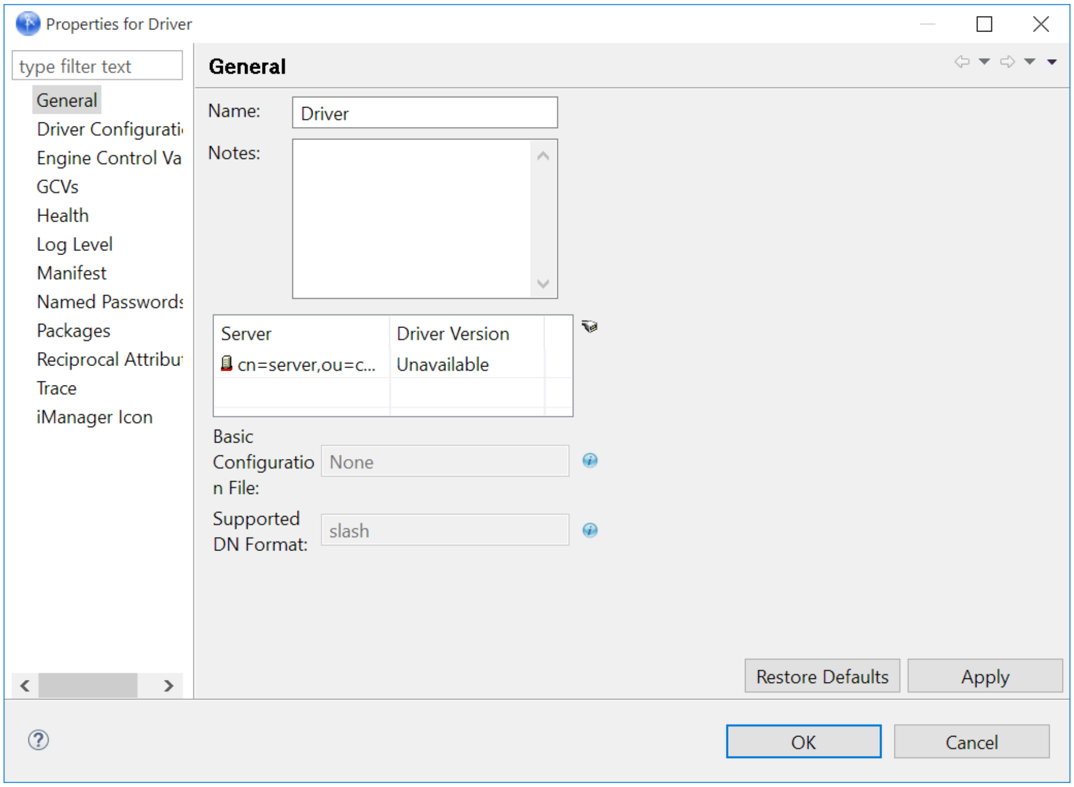
18 After you have installed the driver, you must change a few specific configurations based on your environment. Proceed to "Configuring the Driver Object" on page 41.

For more information, see:

- "Configuring the Remote Loader and Drivers" in the *NetIQ Identity Manager Setup Guide for Linux*
- "Configuring the Remote Loader and Drivers" in the *NetIQ Identity Manager Setup Guide for Windows*

Configuring the Driver Object

After the driver packages are installed, you need to configure the driver before it can run. You should complete the following tasks to configure the driver:



General

Name: Driver

Notes:

Server	Driver Version
cn=server,ou=c...	Unavailable

Basic Configuration File: None

Supported DN Format: slash

- **Configure the driver parameters:** There are many settings that can help you customize and optimize the driver. The settings are divided into categories such as Driver Configuration, Engine Control Values, and Global Configuration Values (GCVs). Although it is important for you to understand all of the settings, your first priority should be to review the “Driver Parameters” on page 94 located on the Driver Configuration page. The Driver Parameters let you configure the publication method and other parameters associated with the Publisher channel.
- **Customize the driver policies and filter:** The driver policies and filter control data flow between the Identity Vault and the application. You should ensure that the policies and filters reflect your business needs. For instructions, see Chapter 5, “Customizing the Driver for RESTful Services,” on page 51.
- **Set Up a Secure HTTPS Connection:** The connection between the driver and the RESTful connected system can be configured to use a secure HTTPS connection rather than an HTTP connection.

After completing the configuration tasks, continue with“Deploying the Driver Object” on page 42.

Deploying the Driver Object

After the driver object is created in Designer, it must be deployed into the Identity Vault.

1. In Designer, open your project.
2. In the Modeler, right-click the driver icon or the driver line, then select **Live > Deploy**.
3. If you are authenticated to the Identity Vault, skip to Step 4; otherwise, specify the following information, then click **OK**.

Field	Description
Host	Specify the IP address or DNS name of the server hosting the Identity Vault.
Username	Specify the DN of the user object used to authenticate to the Identity Vault.
Password	Specify the user’s password.

4 Read the deployment summary, then click **Deploy**.

5 Read the message, then click **OK**.

6 Click **Define Security Equivalence** to assign rights to the driver.

The driver requires rights to objects within the Identity Vault. The Admin user object is most often used to supply these rights. However, you might want to create a DriversUser (for example) and assign security equivalence to that user. Whatever rights that the driver needs to have on the server, the DriversUser object must have the same security rights.

6a Click **Add**, then browse to and select the object with the correct rights.

6b Click **OK** twice.

For more information about defining a Security Equivalent User in objects for drivers in the Identity Vault, see the *NetIQ Identity Manager Security Guide*

7 Click **Exclude Administrative Roles** to exclude users that should not be synchronized.

You should exclude any administrative User objects (for example, Admin and DriversUser) from synchronization.

7a Click **Add**, then browse to and select the user object you want to exclude, then click **OK**.

7b Repeat Step 7a for each object you want to exclude, then click **OK**.

8 Click **OK**.

9 Continue with the next section,“Starting the Driver” on page 43.

Starting the Driver

When a driver is created, it is stopped by default. To make the driver work, you must start the driver. Identity Manager is an event-driven system, so after the driver is started, it won’t do anything until an event occurs. You can use Identity Console or `dxevent` commands to start the driver.

To start the driver using Designer:

1 In Designer, open your project.

2 In the Modeler, right-click the driver icon or the driver line, then select **Live > Start Driver**.

To start the driver using Identity Console:

1 Login to Identity Console,

2 Click **IDM Administration**.

3 Browse to and select the driver set object that contains the driver you want to start.

4 Click the driver set name to access the Driver Set Overview page.

5 Click the Action Menu ⋮ of the driver, then click **Start driver**.

IMPORTANT: When you start the driver for the first time, don't add new users to the Publisher channel until the first polling interval completes because the driver treats all users as existing users and stores them in the change cache without sending them to the Identity Manager engine. It sends the new users to the Identity Manager engine from the next polling interval. Therefore, ensure that new users are added to the Publisher channel after the first polling cycle completes.

Activating the Driver

The Identity Manager driver for REST is part of the Identity Manager Integration Module for Tools. This integration module includes the following drivers:

- Identity Manager driver for Delimited Text
- Identity Manager driver for SOAP
- Identity Manager driver for REST

This integration module requires a separate activation. After purchasing the integration module, you will receive activation details in your NetIQ Customer Center.

If you create a new REST driver in a driver set that already includes an activated driver from this integration module, the new driver inherits the activation from the driver set.

If you create the driver in a driver set that has not been previously activated with this integration module, the driver will run in the evaluation mode for 90 days. You must activate the driver with this integration module during the evaluation period; otherwise, the driver will be disabled.

If driver activation has expired, the trace displays an error message indicating that you need to reactivate the driver to use it. For information on activation, refer to Activating Identity Manager in the *NetIQ Identity Manager Overview and Planning Guide*.

Adding Packages to an Existing Driver

You can add new functionality to an existing driver by adding new packages to it.

1 Right-click the driver, then click **Properties**.

2 Click **Packages**, then upgrade the already installed REST Base package.

   2a Select the package from the list of packages, then click the **Select Operation** cell.

   2b Click **Upgrade** from the drop-down list, then click **Apply**.

   2c Click **OK** to close the Package Management page.

   You can upgrade the Password Synchronization package in a similar way.

3 Click the **Add Packages** icon +.

4 Select the packages to install.

5 (Optional) If you want to see all available packages for the driver, clear the **Show only applicable package versions** option, if you want to see all available packages for the driver, then click **OK**.

   This option is only displayed on drivers. By default, only the packages that can be installed on the selected driver are displayed.

6 Click **Apply** to install all of the packages listed with the Install operation.

7. (Conditional) Fill in the fields with appropriate information to install the package you selected for the driver, then click **Next**.
8. Read the summary of the installation, then click **Finish**.
9. Click **OK** to close the Package Management page after you have reviewed the installed packages.
10. Modify the driver configuration settings. See “Configuring the Driver Object” on page 41.
11. Deploy the driver. See “Deploying the Driver Object” on page 42.
12. Start the driver. See “Starting the Driver” on page 43.
13. Repeat Step 1 through Step 9 for each driver where you want to add the new packages.

4 Upgrading an Existing Driver

The following sections provide information to help you upgrade an existing driver:

- "Supported Upgrade Paths" on page 47
- "Upgrading the Driver" on page 47

Supported Upgrade Paths

You can upgrade from 1.0 version of the REST driver to 1.0.1.1 version.

Upgrading the Driver

The REST driver upgrade process involves upgrading the installed driver packages and updating the driver files.

This section provides general instructions for updating a driver. For information about updating the driver to a specific version, search for that driver patch in the NetIQ Patch Finder Download Page and follow the instructions from the Readme file accompanying the driver patch release.

- "Upgrading the Installed Packages" on page 47
- "Applying the Driver Patch" on page 48

Upgrading the Installed Packages

1 Download the latest available packages.

To configure Designer to automatically read the package updates when a new version of a package is available, click **Windows > Preferences > NetIQ > Package Manager > Online Updates** in Designer. For more information about managing packages, see the *NetIQ Designer for Identity Manager Administration Guide*.

2 Upgrade the installed packages.

2a Open the project containing the driver.

2b Right-click the driver for which you want to upgrade an installed package, then click **Driver > Properties**.

2c Click **Packages**.

If there is a newer version of a package, there is check mark displayed in the Upgrades column.

2d Click **Select Operation** for the package that indicates there is an upgrade available.

2e From the drop-down list, click **Upgrade**.

2f Select the version that you want to upgrade to, then click **OK**.

NOTE: Designer lists all versions available for upgrade.

2g Click **Apply**.

2h (Conditional) Fill in the fields with appropriate information to upgrade the package, then click **Next**.

Depending on which package you selected to upgrade, you must fill in the required information to upgrade the package.

2i Read the summary of the packages that will be installed, then click **Finish**.

2j Review the upgraded package, then click **OK** to close the Package Management page.

For detailed information, see the "Upgrading Installed Packages" in the *NetIQ Designer for Identity Manager Administration Guide*.

Applying the Driver Patch

The driver patch updates the driver files. You can install the patch as a `root` or `non-root` user.

- "Prerequisites" on page 48
- "Applying the Patch as a Root User" on page 48
- "Applying the Patch as a Non-Root User" on page 49

Prerequisites

Before installing the patch, complete the following steps:

1. Take a back-up of the current driver configuration.
2. (Conditional) If the driver is running with the Identity Manager engine, stop the Identity Vault and the driver instance.
3. (Conditional) If the driver is running with a Remote Loader instance, start the Remote Loader instance and the driver instance.
4. In a browser, navigate to the NetIQ Patch Finder Download Page.
5. Under **Patches**, click **Search Patches**.
6. Specify **Identity Manager *nn* REST DRIVER nn** in the search box.
7. Download and unzip the contents of the patch file to a temporary location on your server.

   For example, `IDM45_REST_1001.zip`.

Applying the Patch as a Root User

In a root installation, the driver patch installs the driver files RPMs in the default locations on Linux. On Windows, you need to manually copy the files to the default locations.

1. Update the driver files:
   - **Linux:** To upgrade the existing RPM, log in as `root` and run the following command in a command prompt:

```
rpm -Uvh <Driver Patch File Temporary Location>/linux/netiq-DXMLRESTDrv.rpm
```

For example, `rpm -Uvh <`*`IDM4.5_FanoutAgent_1110.zip`*`>/linux/netiq-DXMLRESTDrv.rpm`

- **Windows:** Navigate to the <*`Extracted Driver Patch File Temporary Location`*>`\windows` folder and copy the following files to `<IdentityManager installation>\NDS\lib` or `<IdentityManager installation>\RemoteLoader\lib` folder:
  - `RESTCommon.jar`
  - `RESTDriverShim.jar`
  - `RESTUtil.jar`

2 (Conditional) Start the Remote Loader instance.

3 (Conditional) Start the REST driver.

Applying the Patch as a Non-Root User

1 Verify that <*`non-root eDirectory location`*>`/rpm` directory exists and contains the file, `_db.000`.

If `_db.000` is not present in this directory, the installation will not succeed.

2 To set the `root` directory to non-`root` eDirectory location, enter the following command in the command prompt:

`ROOTDIR=<`*`non-root eDirectory location`*`>`

This will set the environmental variables to the directory where eDirectory is installed as a non-`root` user.

3 To install the driver files, enter the following command:

```
rpm --dbpath $ROOTDIR/rpm -Uvh --relocate=/usr=$ROOTDIR/opt/novell/
eDirectory --relocate=/etc=$ROOTDIR/etc --relocate=/opt/novell/
eDirectory=$ROOTDIR/opt/novell/eDirectory --relocate=/opt/novell/
dirxml=$ROOTDIR/opt/novell/dirxml --relocate=/var=$ROOTDIR/var --
badreloc --nodeps --replacefiles <rpm-location>
```

For example, to install the REST driver RPM, use this command:

```
rpm --dbpath $ROOTDIR/rpm -Uvh --relocate=/usr=$ROOTDIR/opt/novell/
eDirectory --relocate=/etc=$ROOTDIR/etc --relocate=/opt/novell/
eDirectory=$ROOTDIR/opt/novell/eDirectory --relocate=/opt/novell/
dirxml=$ROOTDIR/opt/novell/dirxml --relocate=/var=$ROOTDIR/var --
badreloc --nodeps --replacefiles /home/user/novell-DXMLRESTDrv.rpm
```

5 Customizing the Driver for RESTful Services

The following sections provide information to help you understand the available customization to make the driver connect to any RESTful service:

Modifying Java Extensions

Use the java extensions to modify a REST request or response before it is submitted or received on the Subscriber or Publisher channels. To create Java extensions, the modifier class file (for example <`SFDocModifier.jar`>) must be available in the path `/opt/novell/eDirectory/lib/dirxml/classes`.

You can modify the following requests and responses using Java extensions:

- Subscriber request document to the connected application.
- Subscriber response document for Identity Manager.
- Publisher request document sent through the Publisher channel to the connected application.
- Publisher response document received through the publisher channel to Identity Manager.

For more information, see Appendix B, "Using Java Extensions," on page 105.

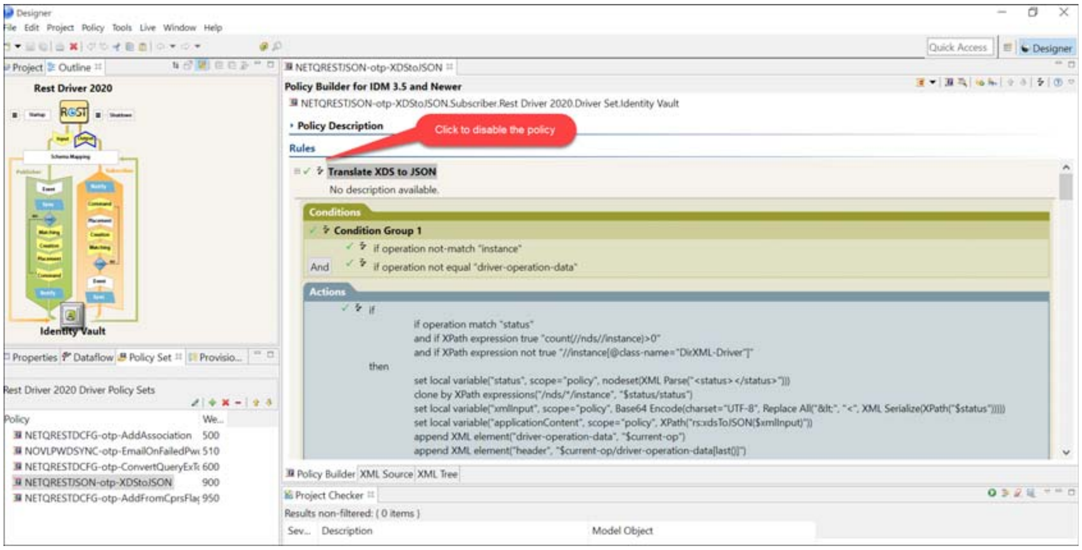
Modifying the JSON/XML Payload

After you install the default JSON package, you can transform the payload generated in the `<driver-operation-data>` to a format supported by your RESTful service. You can modify the JSON/XML payloads that are received in the Subscriber and Publisher channels through conversion policies. The conversion policies modify the request and response documents to a compatible format that is easily comprehended by both Identity Manager and the connected application..

The conversion can be done using any of the following three methods:

- Use the default XDS to JSON conversion policy to transform the payload generated in the `<driver-operation-data>` to a format supported by your RESTful service.
- Create your own XDS to JSON conversion policy, and ensure to keep the payload in `<driver-operation-data>` so that the driver transfers the same to the connected application. The steps to disable an existing policy and add a new policy is shown below:
  1. Select the REST Driver Object in designer.

2. Select **Outline** tab.
3. Select **Output** from the transformation diagram. This shows you all the output transformation policies.
4. Navigate to the **Policy Set** tab and select the XDS-JSON policy (for example, <*NETQRESTJSON-otp-XDStoJSON*>) you want to disable.
5. Click the ✓ icon to disable the policy as shown in the following image.



NETQRESTJSON-otp-XDStoJSON

Policy Builder for IDM 3.5 and Newer

NETQRESTJSON-otp-XDStoJSON.Subscriber.Rest Driver 2020.Driver Set.Identity Vault

Policy Description

Click to disable the policy

Rules

```
Translate XDS to JSON
   No description available.
  Conditions
   Condition Group 1
          if operation not-match "instance"
   And    if operation not equal "driver-operation-data"
  Actions
        if
                 if operation match "status"
                 and if XPath expression true "count(//nds//instance)>0"
                 and if XPath expression not true "//instance[@class-name="DirXML-Driver"]"
         then
                 set local variable("status", scope="policy", nodeset(XML Parse("<status></status>")))
                 clone by XPath expressions("/nds/*/instance", "$status/status")
                 set local variable("xmlInput", scope="policy", Base64 Encode(charset="UTF-8", Replace All("&lt;", "<", XML Serialize(XPath("$status")))))
                 set local variable("applicationContent", scope="policy", XPath("rs:xdsToJSON($xmlInput)"))
                 append XML element("driver-operation-data", "$current-op")
                 append XML element("header", "$current-op/driver-operation-data[last()]")
```

Rest Driver 2020 Driver Policy Sets

Policy	We...
NETQRESTDCFG-otp-AddAssociation	500
NOVLPWDSYNC-otp-EmailOnFailedPw	510
NETQRESTDCFG-otp-ConvertQueryExT	600
NETQRESTJSON-otp-XDStoJSON	900
NETQRESTDCFG-otp-AddFromCprsFla	950

Results non-filtered: (0 items)

Sev...	Description	Model Object

6. Click ✚, in the **Policy Set** section to add the new policy.
7. Name the policy accordingly.
8. Double click the newly created policy, and navigate to **XML Source** tab in the right pane.
9. Paste the new xml content for this policy in the display area.
10. Save all your changes and deploy these policies to Identity Manager.

- Driver shim automatically performs the conversion without any conversion policies.

Using driver-operation-data

You can use the policies to add a new <`driver-operation-data`> element to the Subscriber channel, or submit a new custom created <`driver-operation-data`> element. The <`driver-operation-data`> element is processed irrespective of the configured handlers. For more information, see “Understanding Driver Operation Data” on page 19.

Modifying JSON with Path Expressions

The payload generated in the <driver-operation-data> needs to be transformed into a format supported by your RESTful service. You can modify the JSON payloads that are received in the Subscriber and Publisher channels through conversion policies. The conversion policies modify the request and response documents to a compatible format that is easily comprehended by both

Identity Manager and the connected application. The REST Driver requires JSON inputs to be in a certain format, so in order to help the user to perform operations such as extracting a value, and modifying the data using the following predefined functions that are available.

JSON Path Expressions

JSON Path expressions resemble XPath expressions for XML and provide a way to navigate the JSON structure.

Following are the delimiters used in constructing JSON Path:

Path expression	Description
$	To provide the root object or array.
`.property`	To access the specified property in a parent object.
[n]	To access the n-th element from an array. Indexes are 0-based. To access objects in JSON Array Ex: [id eq P000000] Note: Based on its attribute, we can filter JSON objects contained within an array.
[`property` eq `value`]	Iterates JSON object array and returns object whose `.property` is equal to the specified value.

You can execute the above JSON path expressions using the available functions with examples given below:

1. **getJSONValue:** This function parses the input JSON string and returns the JSON path expressions.

   Example:

   Syntax: String getJsonValue(String jsonstring, String jsonPath)

   Input parameters:

   - jsonstring =

     ```
     "emails": [
     {
     "value": "adrian.stephens@microfocus.com",
     "primary": true
     }
     ],
     "name": {
     "givenName": "Adrian",
     "familyName": "Stephens"
     ```

```
},
```

- jsonPath = `$.name`

Output:

Token Value: `"{"givenName":"Adrian","familyName":"Stephens"}"`

2. **modifyJson:** This function modifies the input JSON string with respect to the given JSON path expressions according to the requirement.

   Example:

   Syntax: public static String modifyJson (String oldjson, String newjson)

   Input parameters:

   - oldjson =

     ```
     "emails": [
     {
     "value": "adrian.stephens@microfocus.com",
     "primary": true
     }
     ],
     "name": {
     "givenName": "Adrian",
     "familyName": "Stephens"
     },
     ```

   - newjson = `{"givenName1":"$.name.givenName"}`

   Output:

   Token Value: `{"givenName1":"Adrian"}`

3. **createJson:** This function helps to create a new JSON with respect to the given JSON path expressions.

   Example:

   Syntax: public static String createJson (String newjson, String oldjson)

   Input parameters:

   - oldjson =

     ```
     "emails": [
     {
     "value": "adrian.stephens@microfocus.com",
     "primary": true
     }
     ],
     "name": {
     "givenName": "Adrian",
     "familyName": "Stephens"
     },
     ```

- newjson =

  `{"$.name.firstname":"abc","$.name.givenName":"$.name.givenName","$.emails[].value":"abc@mf.com"}`

Output:

Token Value:
`"{"emails":[{"value":"abc@mf.com"}],"name":{"firstname":"abc","givenName":"Adrian"}}"`

Following are JSON modifiers functions available in policy:

- JSONCreator (https://www.netiq.com/documentation/identity-manager-developer/driver-developer-kit/rest-driver-developer-docs/com/novell/nds/dirxml/driver/rest/common/JSONCreator.html?)
- JSONParser (https://www.netiq.com/documentation/identity-manager-developer/driver-developer-kit/rest-driver-developer-docs/com/novell/nds/dirxml/driver/rest/common/JSONParser.html?)

Usage of JSON Modifier for Publisher Polling

This section explains the procedure to use JSON modifier with an example:

1. Add the following paths in policy to call json modifier functions:

   xmlns:jc=”https://www.netiq.com/documentation/identity-manager-developer/driver-developer-kit/rest-driver-developer-docs/com/novell/nds/dirxml/driver/rest/common/JSONCreator.html”.

   xmlns:jp="https://www.netiq.com/documentation/identity-manager-developer/driver-developer-kit/rest-driver-developer-docs/com/novell/nds/dirxml/driver/rest/common/JSONParser.html”.
2. Verify that the HTTP response string length is greater than zero.
3. Extract the HTTP response and set it to a local variable.
4. Extract the Resource array from the response using the **getJsonValue** function.
5. Add modification rules for User and Group resources with the help of JSON path expressions.

   Example : { "class-name" : "User","CN": "$.userName", "Surname": "$.name.familyName", "workforceID": "$.id" }
6. Modify the resource according to the modification rules using **modifyJsonArray** function.
7. Use creation Rules to create new json with modified resource array.

   Example: `{"$.results" : "$"}.` Here we are creating new json which has a result attribute and its value is mapped to a modified json array.
8. Call **createJson** function and pass the above creation rules and modified JSON array.
9. Remove existing http response from driver operation data and add new modified json created from **createJson** function.

Following policy illustrates the above steps:

```
<arg-actions>
  <do-if>
    <arg-conditions>
      <and>
        <if-xpath op="true">string-length(./response/value/text())>0</if-
xpath>
      </and>
    </arg-conditions>
    <arg-actions>
      <do-set-local-variable name="httpResponse">
        <arg-string>
          <token-xpath expression="./response/value"/>
        </arg-string>
      </do-set-local-variable>
      <do-if>
        <arg-conditions>
          <and>
            <if-class-name op="equal">User</if-class-name>
          </and>
        </arg-conditions>
        <arg-actions>
          <do-set-local-variable name="modificationRules">
            <arg-string>
             <token-text>{"class-name" : "User","CN": "$.name", "Surname":
"$.lastName", "workforceID": "$.id"}</token-text>
            </arg-string>
          </do-set-local-variable>
          <do-set-local-variable name="responseArray">
            <arg-string>
              <token-xpath expression='jp:getJsonValue($httpResponse,
"$.users")'/>
            </arg-string>
          </do-set-local-variable>
        </arg-actions>
      </do-if>
      <do-if>
        <arg-conditions>
          <and>
            <if-class-name op="equal">Group</if-class-name>
          </and>
        </arg-conditions>
        <arg-actions>
          <do-set-local-variable name="modificationRules">
            <arg-string>
             <token-text>{ "CN" : "$.name", "Description" : "$.id", "class-
name" : "Group" }</token-text>
            </arg-string>
          </do-set-local-variable>
          <do-set-local-variable name="responseArray">
            <arg-string>
              <token-xpath expression='jp:getJsonValue($httpResponse,
"$.groups")'/>
            </arg-string>
          </do-set-local-variable>
        </arg-actions>
        <arg-actions/>
      </do-if>
      <do-set-local-variable name="modifiedArray">
        <arg-string>
          <token-xpath expression="jp:modifyJsonArray($responseArray,
$modificationRules)"/>
        </arg-string>
      </do-set-local-variable>
      <do-set-local-variable name="creationRules">
        <arg-string>
          <token-text xml:space="preserve">{"$.results" : "$"}</token-text>
        </arg-string>
      </do-set-local-variable>
      <do-set-local-variable name="modifiedHttpResponse">
        <arg-string>
          <token-xpath expression="jc:createJson($creationRules,
$modifiedArray)"/>
        </arg-string>
      </do-set-local-variable>
      <do-strip-xpath expression="./response/value/text()"/>
      <do-append-xml-text expression="./response/value">
        <arg-string>
          <token-local-variable name="modifiedHttpResponse"/>
        </arg-string>
      </do-append-xml-text>
      <do-set-local-variable name="xmlInput" notrace="true" scope="policy">
        <arg-string>
          <token-base64-encode charset="UTF-8">
            <token-replace-all regex="&amp;lt;" replace-with="&lt;">
              <token-xml-serialize>
                <token-xpath expression="."/>
              </token-xml-serialize>
            </token-replace-all>
          </token-base64-encode>
        </arg-string>
      </do-set-local-variable>
      <do-set-local-variable name="applicationContent" notrace="true"
scope="policy">
        <arg-string>
          <token-xpath expression="rs:jsonToXDS($xmlInput)"/>
        </arg-string>
      </do-set-local-variable>
      <do-if notrace="true">
        <arg-conditions>
          <and>
            <if-local-variable mode="nocase" name="applicationContent"
op="not-equal"/>
          </and>
        </arg-conditions>
        <arg-actions>
          <do-set-local-variable name="xdscontent" scope="policy">
            <arg-node-set>
              <token-xml-parse>
```

```
                <token-local-variable name="applicationContent"/>
              </token-xml-parse>
            </arg-node-set>
          </do-set-local-variable>
          <do-clone-xpath dest-expression=".." src-expression="$xdscontent/
input/modify"/>
        </arg-actions>
        <arg-actions/>
      </do-if>
      <do-strip-xpath expression="."/>
    </arg-actions>
    <arg-actions/>
  </do-if>
</arg-actions>
```

REST Driver Pagination

REST API pagination is the process that helps to divide the datasets into distinct pages. It is better to break up the dataset into smaller chunks which will make the response quicker and it is easy to handle.

Supported Pagination techniques:

There are three pagination techniques that can be used:

- Offset Pagination
- Cursor Pagination
- Custom Pagination

Offset Pagination

Pagination technique that uses an offset parameter to determine the starting point of the next set of results.

Parameters	Description
Offset	Starting point
Limit	Size of the page
Total results	Response attribute in the dataset that gives the total count or manually enter the total count.

To update the pagination details, perform the following steps:

1. In Identity Console, go to **Drivers**.
2. Select the driver that you want to perform pagination.
3. Navigate to **Configuration**.
4. Click **Driver parameters** drop-down and go to **Publisher Settings**.

5 Under **Publisher Option**, select **Publisher setting** to **Poll Mode**.

6 Under **Optional Header Fields**, select the **Pagination Type** to **Offset Pagination**.

7 Enter the required details in the following field:

   7a Offset

   7b Page Size

   7c Total Count

8 Click **Save**.

Example

For the following SAP Cloud payload, enter the input parameters as given below:

Input Parameters:

- Offset = 1
- Page Size = 2
- Total Count = totalResults (or) 1209

SAP Cloud payload:

```
{
"schemas": [
"urn:ietf:params:scim:api:messages:2.0:ListResponse"
],
"totalResults": 1209,
"itemsPerPage": 100,
"Resources": [
{
"id": "P000000",
"userUuid": "93876fe3-8f0c-4d2c-9db2-caac1adf83e7",
"displayName": "Adrian Stephens",
"emails": [
{
"value": "adrian.stephens@microfocus.com",
"primary": true
}
],
"name": {
"givenName": "Adrian",
"familyName": "Stephens"
},
"urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {}
},
{
"id": "P000016",
"userUuid": "6bb51346-e077-43ef-810c-82bbb64313ec",
"displayName": "Abhishek Prasad",
"emails": [
{
"value": "abhishek.prasad@microfocus.com",
"primary": true
}
],
"name": {
"givenName": "Abhishek",
"familyName": "Prasad"
},
"urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {}
},
{
"id": "P000238",
"userUuid": "6055737e-7e18-4732-a9aa-29b4966a569c",
"userName": "tgarrison",
"displayName": "Tana Garrison",
"emails": [
{
"value": "tgarrison@extfocus.com",
"primary": true
}
],
"name": {
"givenName": "Tana",
"middleName": "s",
"familyName": "Garrison"
},
"urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
"employeeNumber": "1850"
}
}
]
}
```

Cursor Pagination

This uses a cursor parameter to determine the starting point of the next set of results. The cursor can be a unique identifier or a bookmark that points to a specific location in the result set.

The Cursor pagination is further divided into two according to the cursor placement on the dataset:

- Link-Header based Pagination
- Key set/Token Pagination

Link-Header based Pagination

Link-Header based pagination relies on HTTP headers to provide pagination meta data. The HTTP Link entity-header field provides a means for serializing one or more links in HTTP headers.

To update the pagination details, perform the following steps:

1. In Identity Console, go to **Drivers**.
2. Select the driver that you want to perform pagination.
3. Navigate to **Configuration**.
4. Click **Driver parameters** drop-down and go to **Publisher Settings**.
5. Under **Publisher Option**, select **Publisher setting** to **Poll Mode**.

6 Under **Optional Header Fields**, select the **Pagination Type** to **Cursor Pagination**.

7 Select the **Cursor location** to **Response Header**.

8 Enter the required details in the following field:

  8a Link Header Relation

  8b Link Header Value

  8c Absolute URL (Yes/No)

  8d (optional) If it is not an Absolute URL, Specify first Page URL.

  8e Page Size.

9 Click **Save**.

It is further explained by a syntax with an example below:

Syntax: Link: <uri-reference1>; param1=value1; param2="value2", <uri-reference2>; param1=value1; param2="value2"

Parameters	Description
Param	link relation
Value	link relation value

Example:

Enter the following Input Parameter:

- Link Header Relation = rel
- Link Header Value = “next”
- Absolute URL = Yes
- Page Size = 2

Link:

```
<https://api.github.com/repositories/1300192/issues?page=2>; rel="prev",
<https://api.github.com/repositories/1300192/issues?page=4>; rel="next",
<https://api.github.com/repositories/1300192/issues?page=515>; rel="last",
<https://api.github.com/repositories/1300192/issues?page=1>; rel="first".
```

After parsing the link header, it is stored as key value pair as shown below:

```
{ rel:
{ prev: 'https://api.github.com/repositories/1300192/issues?page=2',
next: 'https://api.github.com/repositories/1300192/issues?page=4',
last: 'https://api.github.com/repositories/1300192/issues?page=515',
first: 'https://api.github.com/repositories/1300192/issues?page=1'
}
}
```

Key set/Token pagination

In this technique the cursor is present in the response body.

Parameters	Description
Cursor Key	Attribute in response that contains the cursor
Terminating Key	Attribute in response that indicates the last page
Terminating value	Pagination is done when the Terminating key value matches this value
Page size	Number of records on given page

To update the pagination details, perform the following steps:

1. In Identity Console, go to **Drivers**.
2. Select the driver that you want to perform pagination.
3. Navigate to **Configuration**.
4. Click **Driver parameters** drop-down and go to **Publisher Settings**.
5. Under **Publisher Option**, select **Publisher setting** to **Poll Mode**.
6. Under **Optional Header Fields**, select the **Pagination Type** to **Cursor Pagination**.
7. Select the **Cursor location** to **Response Payload**.
8. Enter the required details in the following field:
   - **8a** Cursor key
   - **8b** Terminating Key
   - **8c** Terminating Value
   - **8d** Absolute URL (Yes/No)
   - **8e** (optional) If it is not an Absolute URL, Specify first Page URL.
   - **8f** Page Size.
9. Click **Save**.

Example:

Enter the following Input Parameters:

- Cursor key = nextPageURL
- Terminating Key = nextPageURL
- Terminating Value = null
- Absolute URL = Yes
- Page Size = 2

Salesforce Payload:

```
{
"currentPageUrl": "/services/data/v23.0/chatter/users",
"nextPageUrl": null,
"previousPageUrl": null,
"users": [
{
"companyName": "Open Text",
"firstName": Tana,
"id": "0055i000004C5CfAAK",
"lastName": "Garrison",
"name": "Tana Garrison",
"type": "User",
"url": "/services/data/v23.0/chatter/users/0055i000004C5CfAAK"
},
{
"companyName": "Open Text",
"firstName": "Adrian",
"id": "0055i000004C5CaAAK",
"lastName": "Stephens",
"name": "Adrian Stephens",
"type": "User",
"url": "/services/data/v23.0/chatter/users/0055i000004C5CaAAK"
},
{
"companyName": "Microfocus",
"firstName": "Tejas",
"id": "0055i000004HQiFAAW",
"lastName": "MP",
"name": "Tejas MP",
"type": "User",
"url": "/services/data/v23.0/chatter/users/0055i000004HQiFAAW"
},
{
"companyName": Microfocus,
"firstName": "Abhishek",
"id": "0055i000004HubrAAC",
"lastName": "Prasad",
"name": "Abhishek Prasad",
"type": "User",
"url": "/services/data/v23.0/chatter/users/0055i000004HubrAAC"
}
]
}
```

Custom Pagination

In some cases, the implementation details can vary depending on the specific inputs used. There is custom pagination in which you must provide the interface that has to be implemented. The implementation can be done by the following methods:

- **init**(String resourceUrl, HashMap<String, String> paginationParm) throws PaginationException;

- **getFirstPageUrl**() throws PaginationException;
- **getNextPageUrl**(String response, HashMap<String, String> responseHeaders) throws PaginationException;

To update the pagination details, perform the following steps:

1. In Identity Console, go to **Drivers**.
2. Select the driver that you want to perform pagination.
3. Navigate to **Configuration**.
4. Click **Driver parameters** drop-down and go to **Publisher Settings**.
5. Under **Publisher Option**, select **Publisher setting** to **Poll Mode**.
6. Under **Optional Header Fields**, select the **Pagination Type** to **Custom Pagination**.
7. Enter the required details in the following field:

   7a Java Class

   7b Init Parameters
8. Click **Save**.

For more details check the following Section Appendix B, “Using Java Extensions,” on page 105 and check the following JavaDocs (https://www.netiq.com/documentation/identity-manager-developer/driver-developer-kit/rest-driver-developer-docs/) as well.

6 Securing Communication

If the remote Web service you are accessing allows HTTPS connections, you can configure the driver to take advantage of this increased security.

IMPORTANT: Only certificates from a Java keystore are accepted. Make sure that the keystore for the certificates is a Java keystore.

The following sections provide instructions for creating a secure connection:

- “Configuring the Publisher Channel” on page 65
- “Configuring the Subscriber Channel” on page 66

Configuring the Publisher Channel

The Publisher channel publishes the information from the RESTful service to the Identity Vault. To establish a secure connection for the Publisher Channel, you need a keystore or a KMO containing a certificate issued by the certificate authority that signed the server’s certificate.

1 Create a server certificate in Identity Console:

1a Under **eDirectory**, click **Certificate Management > Server Certificate Management**.

1b Click Create Server Certificate icon +.

1c Browse to and select the server object where the REST driver is installed.

1d Specify a certificate nickname.

1e Select **Standard** as the creation method, then click **Next**.

1f Click **OK**.

2 Export a self-signed certificate from the certificate authority in eDirectory:

2a Under **eDirectory**, click **Certificate Management > Server Certificate Management**.

2b Select your tree’s certificate authority object.

It is usually found in the Security container and is named something like *TREENAME CA.Security*.

2c Click 🗎 **Export Server Certificate** icon.

2d Uncheck the **Export private key** option.

2e Based on the client to be accessing the Web service, select either **File in binary DER format** or **File in Base64 format** for the certificate, then click **OK**.

If the client uses a Java-based keystore or trust store, then you can choose either format.

2f Click **Save the exported certificate to a file**.

2g Then click **Close**.

3 Import the self-signed certificate into the client’s trust store:

The steps to import the certificate vary depending on the client that connects to the Publisher channel's HTTPS listener. If the client uses a typical Java keystore, you can perform the following steps to create the keystore:

3a Use the keytool executable that is included with any Java JDK.

For more information on keytool, see Keytool - Key and Certificate Management Tool.

3b Enter the following command at a command prompt:

```
keytool -import -file name_of_cert_file -trustcacerts -noprompt
-keystore filename -storepass password
```

For example:

```
keytool -import -file tree_ca_root.b64 -trustcacerts -noprompt -
keystore dirxml.keystore -storepass novell
```

4 Configure the Publisher channel to use the server certificate you created in Step 1 on page 65:

4a In Identity Console, in the **Identity Manager** view, click **IDM Administration**.

4b Locate the driver set containing the REST driver, then click the driver's icon to display the configuration page.

4c Go to **Configuration** > **Driver Parameters**, then go to **Publisher Settings**.

4d In the **KMO name**, specify the certificate nickname you used in Step 1 on page 65.

5 Click **Apply**, then click **OK**.

Configuring the Subscriber Channel

The Subscriber channel sends information from the Identity Vault to the Web service. To establish a secure connection for the Subscriber channel, you need a trust store containing a certificate issued by the certificate authority that signed the server's certificate. See "Configuring the Publisher Channel" on page 65 for an example.

1 Make sure you have a server certificate signed by a certificate authority.

2 Import the certificate into your trust store or create a new trust store by entering the following command at the command prompt:

```
keytool -import -file name_of_cert_file -trustcacerts -noprompt -
keystore filename -storepass password
```

For example:

```
/opt/netiq/common/jre/bin/keytool -import -file tree_ca_root.b64 -
trustcacerts -noprompt -keystore dirxml.keystore -storepass novell
```

For more information on keytool, see Keytool - Key and Certificate Management Tool.

3 Configure the Subscriber channel to use the trust store you created in Step 2:

3a In Identity Console, in the **Identity Manager** view, click **IDM Administration**.

3b Locate the driver set containing the REST driver, then click the driver's icon to display the configuration page.

3c Go to **Configuration** > **Driver Parameters**, then go to **Subscriber Settings**.

3d In the **Keystore File**, specify the path to the trust store you created in Step 2.

4 Click **Apply**, then click **OK**.

7 Managing the Driver

As you work with the REST driver, there are a variety of management tasks you might need to perform, including the following:

- Starting, stopping, and restarting the driver
- Viewing driver version information
- Using Named Passwords to securely store passwords associated with the driver
- Monitoring the driver’s health status
- Backing up the driver
- Inspecting the driver’s cache files
- Viewing the driver’s statistics
- Using the DirXML Command Line utility to perform management tasks through scripts
- Securing the driver and its information

Because these tasks, as well as several others, are common to all Identity Manager drivers, they are included in one reference, the *NetIQ Identity Manager Driver Administration Guide*.

8 Use Case Based Deployment of REST Driver with Connected Applications

You can configure a REST based driver in Identity Manager to connect to multiple REST based external applications. The following section explains the configuration settings that are specific to the connected application. The required parameters to be configured and their sample values are specified which helps you to configure your REST based driver to establish connectivity between Identity Manager and the connected application.

IMPORTANT: The configuration parameters, sample values and examples mentioned in this chapter are for reference purposes only. You must ensure not to use them directly in your production environment.

Sample Deployment of REST Driver for Salesforce

This section explains the specific configuration details required for deploying the driver with Salesforce.

Creating a Connected App for Identity Manager in Salesforce

Salesforce can be integrated with Identity Manager using API's and standard OAuth2.0 protocols. For more information to create a connected app in Salesforce, see Connected Apps section in Salesforce help pages.

Terminologies of Querying Parameters used in Salesforce and Designer

The following table shows some of the naming conventions used in Salesforce and Designer with descriptions which help you to configure the corresponding parameters accordingly. It is recommended to keep the values of these parameters handy to configure the REST driver to connect to Salesforce easily. For more information on the terminologies and conventions of Salesforce, see Connected App and OAuth Terminology.

Salesforce Terminology	Designer Terminology	Description
NA	Access Token URL	The URL of the server used for requesting token access.
Username	User Name	The username to login to Salesforce.
Password	User Password	The password to login to Salesforce.
grant_type	grant_type **NOTE:** To be configured and appears only for **Generate Bearer Token** option.	It is the method used by the application to procure an access token.
Consumer Key	client_id	The client_id is a public identifier for Salesforce.
Server URL	issuer	The authorization server's URL that uses the https protocol.
User	subject	The user’s unique identity for which the access token is being requested.
client_auth_type	client_auth_type	The client's authorization types configured for granting access to the application.
Digital Signature	recipient_keystore	The keystore recipient alias used to look up the digital signature which contains the public key in Salesforce. The following steps explain how to create the `recipient_keystore`. 1. Create the digital signature, refer to Create a Private Key and Self-Signed Digital Certificate. 2. Create a PKCS12 file from combining server key and server certificate, as shown below, **`openssl pkcs12 -inkey <server key> -in <server certificate> -export -out <filename>.pkcs12`** 3. Import the PKCS12 file into the recipient_keystore, as shown below, **`/opt/netiq/common/jre/bin/keytool -importkeystore -srckeystore <filename>.pkcs12 -srcstoretype pkcs12 -destkeystore <recipient keystore>`**
Recipient keystore password	recipient_storepass	Password for the recipient keystore.
Server certificate password	recipient_keypass	Password for the server certificate.

Salesforce Terminology	Designer Terminology	Description
Refresh Token	refresh_token	Refresh Token is a web token to acquire new access tokens when current access tokens expire or become invalid. The authorization server (Salesforce) provides refresh tokens to the Identity Manager to obtain new access token without user interaction in the backend.
Consumer Secret	client_secret	The client secret is used to establish the ownership of the client_id.

Sample Data Flow Between REST Driver and Salesforce

The following operations can be performed on the subscriber channel:

- **Operations performed on a user**
  - **Adding a user**: A user is added in Identity Manager and synced to Salesforce through the REST driver. The details of the user such as, user's first name, last name, contact details, email ID, location, department, user name, initial login password are added and synchronized with Salesforce.

    The REST end point for Salesforce to add a user: `https://<tenant name>.salesforce.com/services/data/<current version>/sobjects/User`

    IMPORTANT: Ensure to replace the variable values in the REST end point URL as per Salesforce specifications. The sample values are shown as follows, and applicable for the REST end point examples mentioned in other sections.

    - *<tenant name>* with *ap16, ap17,* etc.
    - *<current version>* with *v20.0, v20.1*, etc.
    - *<association>* with *salesforce-userid, salesforce-groupid*, etc.
  - **Modifying a user**: If there are any changes made to the user details such as, user's first name, last name, contact details, email ID etc, they will be synchronized with Salesforce.

    The REST end point for Salesforce to modify a user: <`https://<tenant name>.salesforce.com/services/data/<current version>/sobjects/User/<salesforce-userid>`

    NOTE: The user can be disabled in case of separation or termination of their services.
- **Operations performed on public groups**
  - **Adding a group**: A group is added in Identity Manager to manage multiple users with same set of access permissions, rather than managing users individually.

    The REST end point for Salesforce to add a group: <`https://<tenant name>.salesforce.com/services/data/<current version>/sobjects/Group`
  - **Modifying a group**
    - **Adding member to a group**: A member is added to a group based on the user’s role, department and access permissions that the user qualifies for, so that the access permissions for that designated user role are provisioned accordingly.

The REST end point for Salesforce to add a member to a group: <`https://<`*`tenant name`*`>.salesforce.com/services/data/<`*`current version`*`>/sobjects/GroupMember`

  - **Removing member from a group**: A user can be removed from a group if the user's role or designation, or access permissions provided do not qualify a user to belong to that group. This happens in case of a role or designation change of the user, or separation or termination of the user.

  - **Renaming group object**: The group name can be renamed as required.

    The REST end point for Salesforce to rename a group: <`https://<`*`tenant name`*`>.salesforce.com/services/data/<`*`current version`*`>/sobjects/Group/<`*`salesforce-groupid`*`>`

- **Deleting a group**: Duplicate groups, redundant groups, empty groups or groups that are not required can be deleted, and the group members will be moved to another group as required.

  The REST end point for Salesforce to delete a group: <`https://<`*`tenant name`*`>.salesforce.com/services/data/<`*`current version`*`>/sobjects/Group/<`*`salesforce-groupid`*`>`

Creating REST Driver Object for Connecting to Salesforce in Designer

To begin with the configuration, you need to set up the REST driver object in the designer first, and then configure the REST driver with the specific parameters to connect to Salesforce application.

The procedure to set up the REST driver in designer is similar for any connected application. However, configuring the driver to an application depends on the configuration parameters required for that specific application.

The generic steps to set up a driver object in designer is shown from step 1 to step 20, and the configuration parameters specific to Salesforce is mentioned in step 21. If you are familiar with the generic driver object set up, you can choose to skip to Step 21 on page 76 to see the configuration parameters specific to Salesforce application.

1. Open Designer.
2. In the toolbar, click **Help** > **Check for Package Updates**.
3. Click **OK** to update the packages

   or

   Click **OK** if the packages are up-to-date.
4. In the Outline view, right-click the **Package Catalog**.
5. Click **Import Package**.
6. Select any REST driver package

   or

   Click **Select All** to import all of the packages displayed.

   By default, only the base packages are displayed. Deselect **Show Base Packages Only** to display all packages.

7. Click **OK** to import the selected packages, then click **OK** in the successfully imported packages message.
8. In **Designer > Outline** view, open your project.
9. Right click project > **New** > **Identity Vault**, or drag and drop Identity Vault from the Palette to Modeler window.
10. In the Add Server Association screen, select the following field values and click **OK**.
    - Server DN
    - Identity Manager Version
    - Identity Manager Edition

    The Identity Vault Credentials window appears.
11. In Identity Vault Credentials window, enter:

Field	Description
Host	The identity vault hosting machine's IP address
Username	The name of the user, for example, Admin, if the user is an administrator.
Password	The password of the user to login to the identity vault

12. Select **Save Password**, if you want to save your password for easy logins in the future.
13. Click **OK**.

    The Identity Vault and the Driver Set appears in the Modeler window.
14. In the right pane, drag and drop the REST Server from the **Tools** tab to the Modeler.
15. In the Driver Configuration Wizard, select **REST Base** (Contains the base functionality for a driver. You must install a driver base configuration package first).

    NOTE: You can only select one base package.
16. Click **Next**.
17. Select the optional features to install for the REST driver, the options are:
    - REST Default Package
    - REST JSON Package: This package contains the default JSON configurations
    - REST Password Sync: This packages contains the policies that enable the REST driver to synchronize passwords. If you want to synchronize passwords, verify that this option is selected. For more information, see the *NetIQ Identity Manager Password Management Guide*.
18. Click **Next**.

    The package dependencies window appears.
19. (Conditional) Click **OK** to install the package dependency listed.

    NOTE: If there are any dependent packages associated with the selected package, you must install them to proceed.

20 On the Driver Information page, specify a name for the driver, then click **Next**. The Subscriber Options page appears.

21 Select **OAuth 2.0** in the Authentication Method field, as the REST Driver will be configured to connect to Salesforce with **OAuth 2.0** as the authentication method.

22 In the **OAuth2.0 Token Management field,** select the option as required. The available options are:

- **Generate Bearer Token**: The Bearer token is a security token without a signed digital server certificate. Accessing Salesforce with a bearer token may restrict you to execute or perform some operations in Salesforce. You may configure a bearer token if you need access to perform minimal operations.

  To configure REST Driver using **Generate Bearer Token**, see "Configuring REST Driver with Bearer Token" on page 76.

- **Generate JWT Token**: A JSON Web Token (JWT) is signed by a digital server certificate for enhanced security to connect to Salesforce. You can perform all operations without any access restrictions when you configure a JWT Token. Select this option if you want to configure the REST driver with a JWT Token.

  To configure REST Driver using **Generate JWT Token**, see "Configuring REST Driver with JWT Token" on page 79

- **Enter Bearer Token**: Select **Enter Bearer Token** if you already have one available or created by an external application.

  To configure REST Driver using an available bearer token, see "Configuring REST Driver with an Available Bearer Token" on page 81

NOTE: Configuring a **JWT Token** is preferred over a **Bearer Token**, as it is more secured with a digital server certificate, and enables you to perform all operations without any access restrictions.

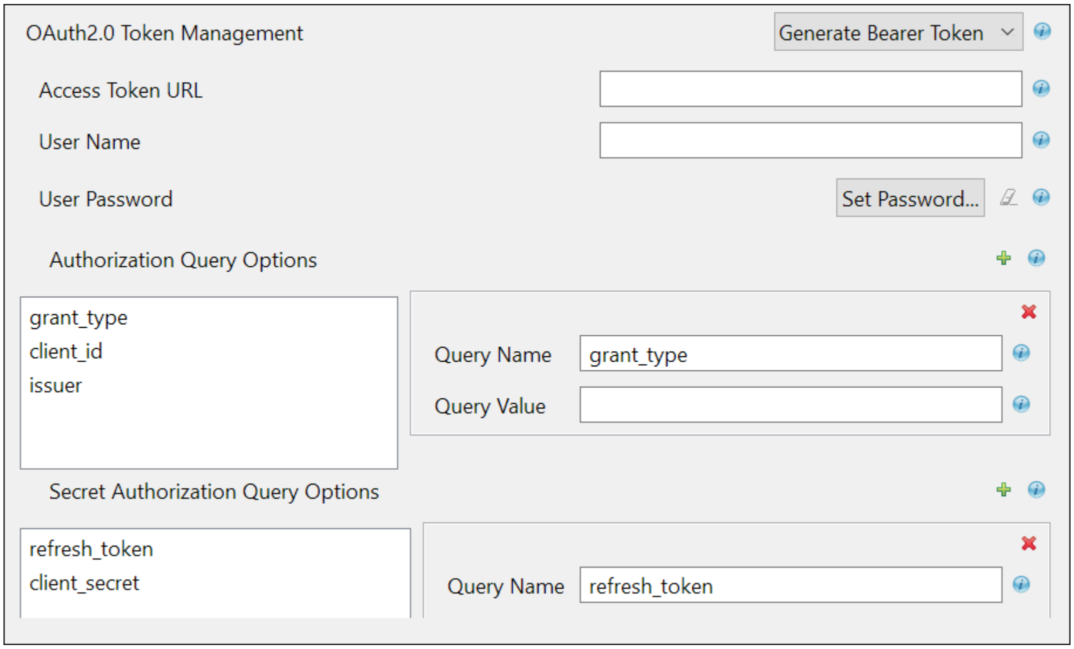
23 There are no publisher options to be specified in the subsequent **Publisher Options** screen, as the publisher channel is not supported for Salesforce application, hence click **Next**.

24 Review the summary of tasks that will be completed to create the driver, then click **Finish**. The configured driver appears in the designer screen.

25 After competing the above steps, refer to "Configuring Resources to Synchronize" on page 83, to configure the resources to synchronize with Salesforce.

Configuring REST Driver with Bearer Token

Generate Bearer Token is an access token issued by servers (Salesforce) to achieve multi-server authentication.



OAuth2.0 Token Management: Generate Bearer Token

Access Token URL

User Name

User Password: Set Password...

Authorization Query Options

grant_type
client_id
issuer

Query Name: grant_type

Query Value

Secret Authorization Query Options

refresh_token
client_secret

Query Name: refresh_token

If you select **Generate Bearer Token**, the following fields appear. Enter the values as shown in the following table.

IMPORTANT: For any operation performed on the Salesforce application using OAuth 2.0, an access token is sent for authorization of the user from Salesforce. The access token expires post the session idle time set for Salesforce, or in case of a system restart. The session idle time for access token expiry is set to 4 hours in the Salesforce application by default. However, the session idle time is configurable as per your requirement. Salesforce displays Unauthozied Access error or an Invalid Session error for any request initiated with an expired access token. The presence of a refresh token helps to re-establish the failed session internally by generating a new access token without the user having to log in again.

NOTE: The * indicates mandatory fields and ^ indicates non mandatory fields.

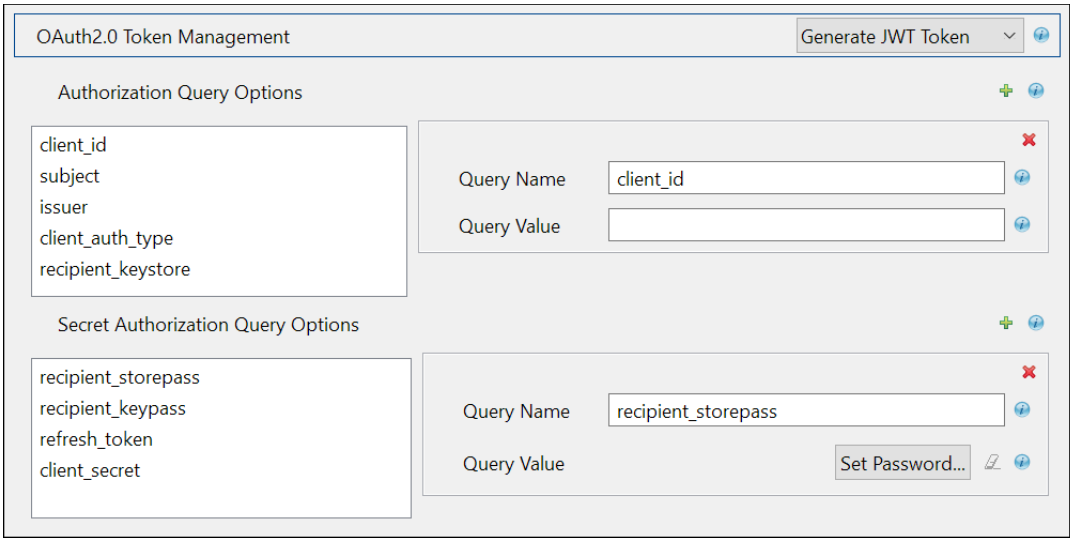
Field	Sample Field Value
Access Token URL	*`<https://login.salesforce.com/services/oauth2/token>`*

Field	Sample Field Value
Authorization Query Options	◆ **`grant_type`**: `password` ◆ **`client_id`**: *`<3MVG97quAmFZJfVwk3ylU.8elhRYBqG9h25m3TWewozjKnFIY0HrhOEJl7LMET9HHocaHnTB1k04kophr1CgW>`* ◆ **`issuer`**: *`<https://login.salesforce.com>`* ◆ **`username`**: *`<username to login to Salesforce>`* **NOTE:** In case of a driver upgrade, the **`issuer`** field does not auto populate the earlier configured value. You must enter the issuer field manually.
Secret Authorization Query Options: These parameters are set to configure a refresh token. Though not mandatory, if configured the set refresh token value is not overridden with the new value when the access token expires. This may cause login issues until the new refresh token is added.	◆ **`refresh_token`**`^`: `5Aep861Xq7VoDavIt6UxKW62EAmfy0hKFv1T_X8yhb9PRQWtsOCrr97CYDrVasefykdl_f.DTVaJGKxjmz50XjQ` ◆ **`client_secret`**`^`: `E734505442694ECD0156D83F965B42C0F07601BB8BFDCA9879420C1FF23C8A87` **IMPORTANT:** When you upgrade the driver the earlier configured **`client_secret`** value that appears in the Authorization Query options is migrated, but the field **`client_secret`** in Secret Auth query options will be blank. Hence it is recommended to delete the migrated client secret from the Authorization query options and manually enter it in the Secret Authorization Query options section as the **`client_secret`**. ◆ **`password`**: *`<password to login to Salesforce>`*
Truststore File: The path and the name of the keystore file that contains the trusted certificates for the remote server to achieve SSL handshake. **IMPORTANT:** For **Generate Bearer Token** add the public certificate to `cacerts`, present in `/opt/netiq/common/jre/lib/security/path`. For more information on securing communication using Truststore file see, "Configuring the Subscriber Channel" on page 66	*`</home/username/SFDer.jks>`* **NOTE:** Create the truststore file in `.jks` format for Salesforce.

Field	Sample Field Value
Set mutual authentication parameters^: Select Show if you want to set mutual authentication information.	- Keystore file: Specify the path and the name of the keystore file that contains the trusted certificates for the remote server to provide mutual authentication. For example, `C:\security\keystore`. Leave this field blank when mutual authentication is not used. - Keystore password: Specify the password for the keystore file. Leave this field blank when mutual authentication is not used.
Http Connection Timeout^: Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces.	*<307 408 503 504>*, and specify the HTTP connection time out value. The driver waits for the time specified and terminates the HTTP connection. The timeout value must be greater than 0.
Proxy host and port^: Specify the host address and the host post when a proxy host and port are used.	192.168.0.0:port. Choose an unused port number on your server. Otherwise, leave this field blank.
Set proxy authentication parameters^ Defaults to Hide. To specify the values select Show.	- User Name: *<proxy username>* - Password: *<password>*
HTTP errors to retry^: Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces.	*<307 408 503 504>*
Base URL for REST Resources*: The URL to which the endpoint paths of Salesforce are appended.	*<https://ap16.salesforce.com/>*

Configuring REST Driver with JWT Token

The JSON Web token is an access request token in the JSON Web Token (JWT) format. It is an encrypted data string consisting of a header, payload, and a signature, and is used to transfer authorization data in client-server applications to authenticate the identity of the resource.



OAuth2.0 Token Management: Generate JWT Token

Authorization Query Options

client_id
subject
issuer
client_auth_type
recipient_keystore

Query Name: client_id

Query Value:

Secret Authorization Query Options

recipient_storepass
recipient_keypass
refresh_token
client_secret

Query Name: recipient_storepass

Query Value: Set Password...

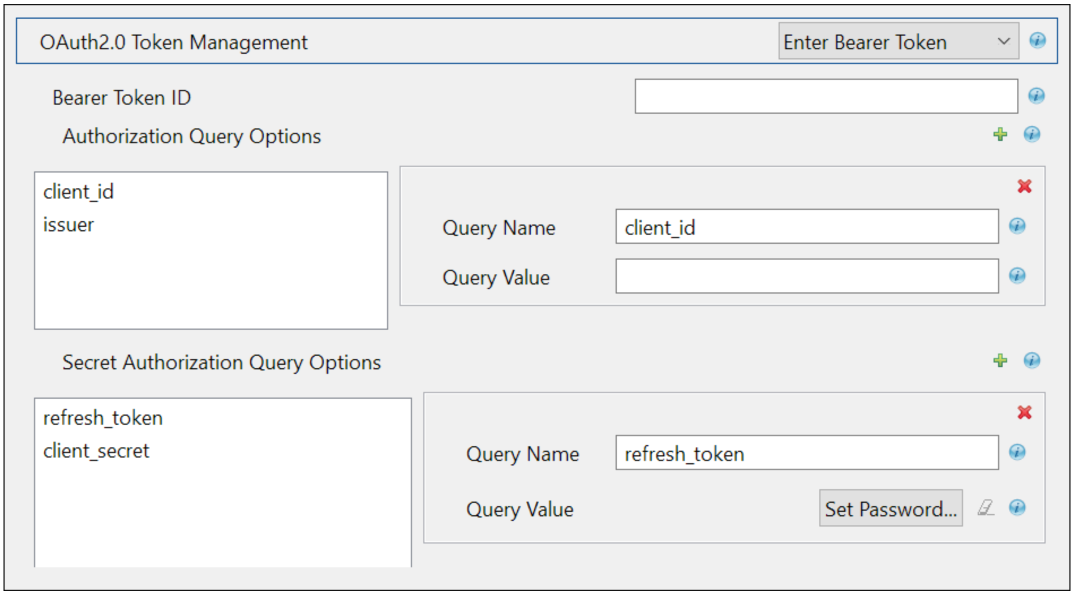
If you select **Generate JWT Token**, the following fields appear:

Field	Sample Field Value
Access Token URL	*<https://login.salesforce.com/services/oauth2/token>*
Authorization Query Options	◆ **client_id**: *<3MVG97quAmFZJfVwk3ylU.8elhRYBqG9h25m3TWewozjKnFIY0HrhOEJl7LMET9HHocaHnTB1k04kophr1CgW>* ◆ **subject**:*<username@microfocus.com>* ◆ **issuer**: *<https://login.salesforce.com>* ◆ **client_auth_type**: private_key_jwt ◆ **recipient_keystore**: *</Soft/Certs/recipient.jks>* ◆ **username**: *<username to login to Salesforce>*
Secret Authorization Query Options: These parameters are set to configure a refresh token. Though not mandatory, if configured the set refresh token value is not overridden with the new value when the access token expires. This may cause login issues until the new refresh token is added.	◆ **recipient_storepass**: *<novell>* ◆ **recipient_keypass**: *<novell>* ◆ **refresh_token**^: 5Aep861Xq7VoDavIt6UxKW62EAmfy0hKFv1T_X8yhb9PRQWtsOCrr97CYDrVasefykdl_f.DTVaJGKxjmz50XjQ ◆ **client_secret**^: E734505442694ECD0156D83F965B42C0F07601BB8BFDCA9879420C1FF23C8A87 **IMPORTANT:** When you upgrade the driver the earlier configured **client_secret** value that appears in the Authorization Query options is migrated, but the field **client_secret** in Secret Auth query options will be blank. Hence it is recommended to delete the migrated client secret from the Authorization Query Options section and enter it manually in the Secret Authorization Query Options section as the **client_secret**. ◆ **password**: *<password to login to Salesforce>*
Truststore File: The path and the name of the keystore file that contains the trusted certificates for the remote server to achieve SSL handshake. For **Generate Bearer Token** add the public certificate to cacerts, present in /opt/netiq/common/jre/lib/security/path. For more information on securing communication using Truststore file see, "Configuring the Subscriber Channel" on page 66	*</home/username/SFDer.jks>* **NOTE:** Create the truststore file in .jks format for Salesforce.

Field	Sample Field Value
Set mutual authentication parameters^: Select Show if you want to set mutual authentication information.	◆ Keystore file: Specify the path and the name of the keystore file that contains the trusted certificates for the remote server to provide mutual authentication. For example, `C:\security\keystore`. Leave this field blank when mutual authentication is not used. ◆ Keystore password: Specify the password for the keystore file. Leave this field blank when mutual authentication is not used.
Http Connection Timeout^: Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces.	*<307 408 503 504>*, and specify the HTTP connection time out value. The driver waits for the time specified and terminates the HTTP connection. The timeout value must be greater than 0.
Proxy host and port^: Specify the host address and the host post when a proxy host and port are used.	192.168.0.0:port. Choose an unused port number on your server. Otherwise, leave this field blank.
Set proxy authentication parameters^ Defaults to Hide. To specify the values select Show.	◆ User Name: <*proxy username*> ◆ Password: <*password*>
HTTP errors to retry^: Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces.	*<307 408 503 504>*
Base URL for REST Resources*: The URL to which the endpoint paths of Salesforce are appended.	For example: <*https://ap16.salesforce.com/*>

Configuring REST Driver with an Available Bearer Token

Select Enter Bearer Token if you already have one available or created by an external application. It is mandatory to configure `refresh_token` and `client_secret` if you select Enter Bearer Token.



OAuth2.0 Token Management: Enter Bearer Token

Bearer Token ID:

Authorization Query Options

client_id
issuer

Query Name: client_id

Query Value:

Secret Authorization Query Options

refresh_token
client_secret

Query Name: refresh_token

Query Value: Set Password...

Field	Sample Field Value
Authorization Query Options	◆ **`client_id`**: *`<3MVG97quAmFZJfVwk3ylU.8elhRYBqG9h25m3TWewozjKnFIY0HrhOEJl7LMET9HHocaHnTB1k04kophr1CgW>`* ◆ **`issuer`**: *`<https://login.salesforce.com>`*
Secret Authorization Query Options: These parameters are set to configure a refresh token. Though not mandatory, if configured the set refresh token value is not overridden with the new value when the access token expires. This may cause login issues until the new refresh token is added.	◆ **`refresh_token`***: `5Aep861Xq7VoDavIt6UxKW62EAmfy0hKFv1T_X8yhb9PRQWtsOCrr97CYDrVasefykdl_f.DTVaJGKxjmz50XjQ` ◆ **`client_secret`***: `E734505442694ECD0156D83F965B42C0F07601BB8BFDCA9879420C1FF23C8A87` **IMPORTANT:** When you upgrade the driver the earlier configured **`client_secret`** value that appears in the Authorization Query options is migrated, but the field **`client_secret`** in Secret Auth query options will be blank. Hence it is recommended to delete the migrated client secret from the Authorization Query Options section and enter it manually in the Secret Authorization Query Options section as the **`client_secret`**.
Truststore File: The path and the name of the keystore file that contains the trusted certificates for the remote server to achieve SSL handshake. For Generate Bearer Token add the public certificate to `cacerts`, present in `/opt/netiq/common/jre/lib/security/path`. For more information on securing communication using Truststore file see, "Configuring the Subscriber Channel" on page 66	*`</home/username/SFDer.jks>`* **NOTE:** Create the truststore file in `.jks` format for Salesforce.
Set mutual authentication parameters^: Select Show if you want to set mutual authentication information.	◆ Keystore file: Specify the path and the name of the keystore file that contains the trusted certificates for the remote server to provide mutual authentication. For example, `C:\security\keystore`. Leave this field blank when mutual authentication is not used. ◆ Keystore password: Specify the password for the keystore file. Leave this field blank when mutual authentication is not used.
Http Connection Timeout^: Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces.	*`<307 408 503 504>`*, and specify the HTTP connection time out value. The driver waits for the time specified and terminates the HTTP connection. The timeout value must be greater than 0.
Proxy host and port^: Specify the host address and the host post when a proxy host and port are used.	192.168.0.0:port. Choose an unused port number on your server. Otherwise, leave this field blank.

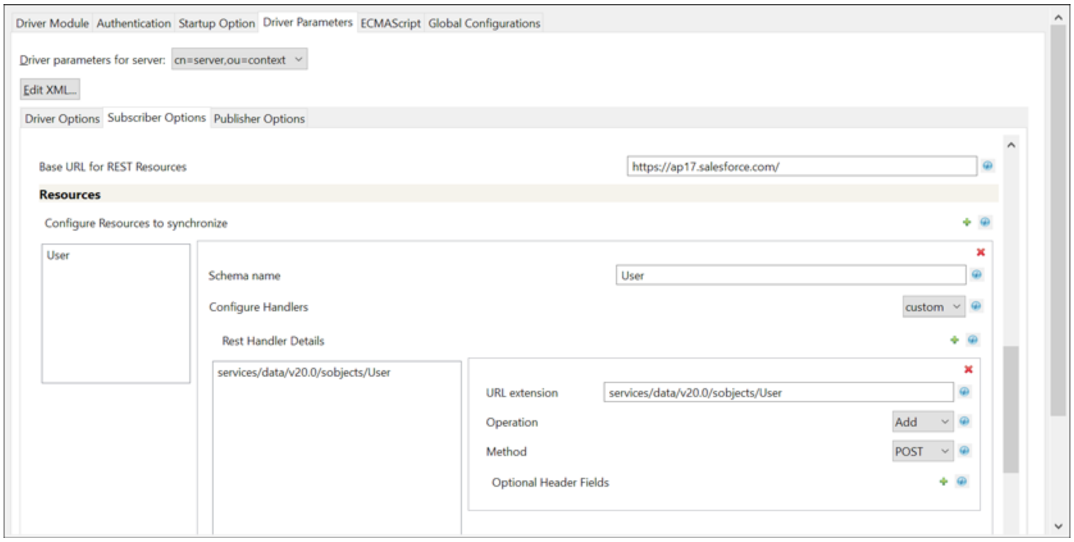
Field	Sample Field Value
Set proxy authentication parameters^ Defaults to Hide. To specify the values select Show.	- User Name: <*proxy username*> - Password: <*password*>
HTTP errors to retry^: Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces.	*<307 408 503 504>*
Base URL for REST Resources*: The URL to which the endpoint paths of Salesforce are appended.	For example: <*https://ap16.salesforce.com/*>

Configuring Resources to Synchronize

Resources are configured to perform operations with the specified method of operation. These operations and methods are appended to the base URL specified to perform the required operation in Salesforce.

The steps to configure resources are as follows:

1. In Designer, double-click the connector.
2. In the Properties screen that appears, select **Driver Configuration** from the left pane.
3. In the Driver Configuration screen, select **Driver Parameters** tab.
4. Select **Subscriber Options**. The Resources section appears.
5. Add **Configure Resource to Synchronize** instance, by clicking the ✚ icon.



Driver parameters for server: cn=server,ou=context

Base URL for REST Resources: https://ap17.salesforce.com/

User

Schema name: User

Configure Handlers: custom

services/data/v20.0/sobjects/User

URL extension: services/data/v20.0/sobjects/User

Operation: Add

Method: POST

6. Provide a schema name, for example *<User>*, as shown in the image to configure handlers.
7. Set the Configure Handlers to **Custom**, and click **instance_1** to add the schema name.
8. Add a **Rest Handler Details** instance by clicking the ✚ icon, and specify the details as shown in the following table:

Field	Description	Sample value
URL extension	Specify the customized URL extension to be appended to the base URL (the object entity on which the action is to be performed), in this case the object entity is `User`.	`services/data/<v20.0>/sobjects/User`
Operation	Specify the type of operation to be performed on the object entity.	The options are: - **Add**: to add the object entity - **Modify**: to modify a object entity - **Query**: to fetch details about the object entity - **Delete**: to delete the object entity
Method	Specify the method for the selected operation	The options are: - **GET**: the method to perform a fetch operation - **POST**: the method to perform a create or add operation - **PUT**: the method to update - **PATCH**: the method to modify - **DELETE**: the method to delete

Example: For adding a user, enter **`services/data/<v20.0>/sobjects/User`** as the URL extension, set the Operation as **ADD**, and Method as **POST**. This URL extension is appended to the base URL, so that the URL generated to perform the add user operation is **`https://<ap17>.salesforce.com/services/data/<v20.0>/sobjects/User`**

NOTE: Ensure to replace the variable values in the URL as required.

9 Click **OK**.

Sample Configuration of Handlers

REST Handler Details	Sample URL Extension	Operation	Method	REST end point URL
Add user	`services/data/<v20.0>/sobjects/User`	Add	POST	`<https://<tenant name>.salesforce.com/services/data/<v20.0>/sobjects/User`
Modify user	`services/data/<v20.0>/sobjects/User/<association>`	Modify	PATCH	`<https://<tenant name>.salesforce.com/services/data/<v20.0>/sobjects/User/<association>`

REST Handler Details	Sample URL Extension	Operation	Method	REST end point URL
Add group	services/data/<*v20.0*>/sobjects/Group	Add	POST	<https://<*tenant name*>.salesforce.com/services/data/<*v20.0*>/sobjects/Group
Modify group	services/data/<*v20.0*>/sobjects/<association>	Add	PATCH	<https://<*tenant name*>.salesforce.com/services/data/<*v20.0*>/sobjects/<association>
Delete group	services/data/<*v20.0*>/sobjects/Group/<*association*>	Delete	DELETE	<https://<*tenant name*>.salesforce.com/services/data/<*v20.0*>/sobjects/Group/<*association*>

Customizing REST Driver Options for Salesforce

This section explains about the policies that can be customized for transforming generic JSON data to Salesforce specific JSON data. The policies that are based on certain rules, perform several actions which can also be customized as required. The REST driver performs multiple transformation operations for any REST based connected system. An example for Salesforce transformation policy is shown below.

IMPORTANT: The below mentioned policies are examples and must be used for reference purposes only. You must ensure not to use them directly in your production environment.

1. Select the REST Driver Object in designer.
2. Select **Outline** tab.
3. Select **Input** from the transformation diagram. This shows you all the input transformation policies.
4. Select the **NETQRESTDCFG-itp-AddAssociation** policy from the **Policy Set** tab, and disable all the rules that are present.
5. Create a new policy by clicking ➕ to add a new input transformation policy, and name it as **NETQREST-itp-UpdateAssociation**. This policy will add the association to the new User or Group, and must be imported into the Input Transformation on Publisher Channel.
6. After the policy is added, navigate to the **XML Source** tab.
7. Copy the following content into the display area.

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE policy PUBLIC "policy-
builder-dtd"
"C:\netiq\idm\apps\Designer\plugins\com.novell.idm.policybuilder_4.0.0
.202002050656\DTD\dirxmlscript4.8.dtd"><policy xmlns:es="http://
www.novell.com/nxsl/ecmascript">
      <rule>
            <description>Check For Association - ADD</description>
            <conditions>
                  <and>
                        <if-operation op="equal">status</if-operation>
                        <if-xpath op="true">driver-operation-data/
response/url-token[@association]</if-xpath>
                        <if-xpath op="not-true">driver-operation-
data[@command='modify']</if-xpath>
                  </and>
            </conditions>
            <actions>
                  <do-set-local-variable name="vl-response">
                        <arg-string>
                              <token-xpath 
expression=".[@level='success']/driver-operation-data[@command='add']/
response/value[last()]/text()"/>
                        </arg-string>
                  </do-set-local-variable>
                  <do-set-local-variable name="vl-association">
                        <arg-node-set>
                              <token-split delimiter='"'>
                                    <token-local-variable name="vl-
response"/>
                              </token-split>
                        </arg-node-set>
                  </do-set-local-variable>
                  <do-add-association>
                        <arg-dn>
                             <token-xpath expression="driver-operation-
data/@dest-dn"/>
                        </arg-dn>
                        <arg-association>
                              <token-xpath expression="$vl-
association[4]"/>
                        </arg-association>
                  </do-add-association>
            </actions>
      </rule>
</policy>
```

8. Click the **Policy Builder** tab. The new policy screen appears.

9. Save all your changes.

10. Similarly, customize the **Output** transformation policies by selecting **Output** in the transformation diagram.

11. Add the following three rules to the **NETQRESTJSON-otp-XDStoJSON** policy xml, to appear above the **Translate XDS to JSON** rule.

```
<rule>
            <description>Remove association and password</description>
            <comment xml:space="preserve">Remove association and 
password</comment>
            <conditions>
                  <and>
                        <if-operation mode="nocase" op="equal">add</if-
operation>
                  </and>
            </conditions>
            <actions>
                  <do-strip-xpath expression="association"/>
                  <do-strip-xpath expression="operation-data/password-
subscribe-status/association"/>
                  <do-strip-xpath expression="password"/>
            </actions>
      </rule>
      <rule>
            <description>populate mandatory attributes</description>
            <comment xml:space="preserve">populate mandatory 
attributes</comment>
            <conditions>
                  <and>
                        <if-operation mode="nocase" op="equal">add</if-
operation>
                        <if-class-name mode="nocase" op="equal">User</
if-class-name>
                  </and>
            </conditions>
            <actions>
                  <do-add-dest-attr-value class-name="User" 
name="TimeZoneSidKey">
                        <arg-value type="string">
                              <token-text xml:space="preserve">Asia/
Kolkata</token-text>
                        </arg-value>
                  </do-add-dest-attr-value>
                  <do-add-dest-attr-value class-name="User" 
name="LocaleSidKey">
                        <arg-value type="string">
                              <token-text xml:space="preserve">en_US</
token-text>
                        </arg-value>
                  </do-add-dest-attr-value>
                  <do-add-dest-attr-value class-name="User" 
name="EmailEncodingKey">
                        <arg-value type="string">
                             <token-text xml:space="preserve">ISO-8859-
1</token-text>
                        </arg-value>
                  </do-add-dest-attr-value>
                  <do-add-dest-attr-value class-name="User" 
name="ProfileId">
                        <arg-value type="string">
                              <token-text 
```

```
xml:space="preserve">00e2v000004F2XO</token-text>
                        </arg-value>
                  </do-add-dest-attr-value>
                  <do-add-dest-attr-value class-name="User"
name="LanguageLocaleKey">
                        <arg-value type="string">
                              <token-text xml:space="preserve">en_US</
token-text>
                        </arg-value>
                  </do-add-dest-attr-value>
            </actions>
      </rule>
      <rule>
            <description>remove attrs from event</description>
            <comment xml:space="preserve">remove attrs from event</
comment>
            <conditions>
                  <and>
                        <if-operation mode="nocase" op="equal">modify</
if-operation>
                  </and>
            </conditions>
            <actions>
                  <do-strip-xpath expression="modify-attr/remove-value/
value"/>
                 <do-strip-xpath expression="modify-attr/remove-value"/
>
            </actions>
      </rule>
```

12 Select **NETQRESTJSON-otp-XDStoJSON** policy in the **Policy** window to create a new policy below it.

13 Create a new policy by clicking + in the **Policy** window, and name it **NETQREST-otp-datatransformation**.

14 Double click the newly created **NETQREST-otp-datatransformation** policy, and navigate to **XML Source** tab in the right pane.

15 Paste the following xml content for this policy in the display area.

```
<?xml version="1.0" encoding="UTF-8"?><policy>
      <rule>
            <description>remove array identifiers</description>
            <comment xml:space="preserve">remove array identifiers</
comment>
            <conditions>
                  <and>
                        <if-xpath op="true">../driver-operation-data/
request/value</if-xpath>
                  </and>
            </conditions>
            <actions>
                  <do-set-local-variable name="valueattribute"
scope="policy">
                        <arg-string>
                              <token-xpath expression="request/value"/>
                        </arg-string>
                  </do-set-local-variable>
                  <do-strip-xpath expression="request/value"/>
                  <do-append-xml-element expression="request"
name="value"/>
                  <do-append-xml-text expression="request/value"
notrace="true">
                        <arg-string>
                             <token-replace-all regex='\"\}\,' replace-
with='",'>
                                    <token-replace-all regex="\}\}"
replace-with="}">
                                          <token-replace-all
regex='\{\"add\"\:' replace-with="">
                                                <token-replace-all
regex="\]" replace-with="">
                                                    <token-replace-all
regex="\[" replace-with="">
                                                          <token-local-
variable name="valueattribute"/>
                                                   </token-replace-all>
                                                </token-replace-all>
                                          </token-replace-all>
                                    </token-replace-all>
                              </token-replace-all>
                        </arg-string>
                  </do-append-xml-text>
            </actions>
      </rule>
</policy>
```

16 Save all your changes and deploy these policies to Identity Manager. This customized REST driver will manage Salesforce system for all the use cases mentioned above.

9 Troubleshooting the Driver

You can log Identity Manager events by using the Event Auditing Service. Using this service in combination with the driver log level setting provides you with tracking control at a very granular level. For more information, see the *Administrator Guide to NetIQ Identity Reporting*.

This section contains the following information on error messages:

- "Hidden JSON Content in Output Transformation Policy Channels" on page 91
- "REST Driver Is Unable to Sync Configured Parameters and Passwords While Upgrading" on page 91
- "Driver Shim Errors" on page 92
- "Troubleshooting Driver Processes" on page 92
- "Driver Reports an Error When a Password or an Attribute Value Contains the < Character" on page 92

Hidden JSON Content in Output Transformation Policy Channels

For security reasons, the content of JSON in the traces are hidden by default. This is done as there may be sensitive information and sensitive attribute values present in the JSON traces. This occurs due to the presence of `Is sensitive` attribute in the output transformation policy channel which suppresses the JSON content.

To troubleshoot and see the hidden JSON content, you must remove the `Is` sensitive attribute. To disable this feature, you should not upgrade the `NETQRESTJSON` package which is optional.

REST Driver Is Unable to Sync Configured Parameters and Passwords While Upgrading

While upgrading the REST driver, the configured parameters and passwords might not appear as configured earlier. You must verify all the configuration screens for non-synced parameters and passwords and enter them manually if required.

Driver Shim Errors

The following errors might occur in the core driver shim. Error messages that contain a numerical code can have various messages, depending on the application or Web service.

- "Issues with commons-codec-1.3.jar" on page 92

Issues with commons-codec-1.3.jar

Explanation: The driver initialization fails if installed on Remote Loader set up.

Possible Cause: Unsupported version of commons-codec-1.3.jar.

Action: Replace the commons-codec-1.3.jar file with the latest version of commons-codec-1.6.jar shipped along with the driver packages.

Troubleshooting Driver Processes

Viewing driver processes is necessary to analyze unexpected behavior. To view the driver processing events, use DSTrace. You should only use it during testing and troubleshooting the driver. Running DSTrace while the drivers are in production increases the utilization on the Identity Manager server and can cause events to process very slowly. For more information, see "Viewing Identity Manager Processes" in the *NetIQ Identity Manager Driver Administration Guide*.

Driver Reports an Error When a Password or an Attribute Value Contains the < Character

Issue: The JSON converter does not accept the "<" character. However, this issue does not occur with the ">" character:

Workaround: No workaround is available.

A Driver Properties

This section provides information about the Driver Configuration and Global Configuration Values properties for the REST driver. These are the only unique properties for drivers. All other driver properties (Named Password, Engine Control Values, Log Level, and so forth) are common to all drivers. Refer to "Driver Properties" in the *NetIQ Identity Manager Driver Administration Guide* for information about the common properties.

- "Driver Configuration" on page 93
- "Global Configuration Values" on page 102

Driver Configuration

In Identity Console:

1. Click the **IDM Administration** tile.
2. Select the driver set that contains the driver whose properties you want to edit.
3. Click the driver icon to display the driver's properties page.

In Designer:

1. Open a project in the Modeler.
2. Right-click the driver icon or line, then select click **Properties > Driver Configuration.**

The Driver Configuration options are divided into the following sections:

- "Driver Module" on page 93
- "Authentication" on page 94
- "Startup Option" on page 94
- "Driver Parameters" on page 94
- "ECMAScript" on page 102
- "Global Configuration" on page 102

Driver Module

The driver module changes the driver from running locally to running remotely or the reverse.

Java: Use this option to specify the name of the Java class that is instantiated for the shim component of the driver. This class can be located in the `classes` directory as a class file, or in the `lib` directory as a `.jar` file. If this option is selected, the driver is running locally. Select this option to run the driver locally.

The Java class name is: `com.novell.nds.dirxml.driver.rest.RESTDriverShim`

Native: This option is not used with the REST driver.

Connect to Remote Loader: Used when the driver is connecting remotely to the connected system. Designer includes two suboptions:

- Remote Loader Client Configuration for Documentation: Includes information on the Remote Loader client configuration when Designer generates documentation for the driver.
- Driver Object Password: Specifies a password for the Driver object. If you are using the Remote Loader, you must enter a password on this page. Otherwise, the remote driver does not run. The Remote Loader uses this password to authenticate itself to the remote driver shim.

Name: Displays the java class name.

Driver Object Password: Use this option to set a password for the driver object. If you are using the Remote Loader, you must enter a password on this page or the remote driver does not run. This password is used by the Remote Loader to authenticate itself to the remote driver shim.

Authentication

The authentication section describes the parameters required for authentication to the connected system. This section is not applicable for the Identity Manager driver for REST. The authentication method for REST driver is Anonymous, Basic or OAuth2.0.

Startup Option

The Startup Option section allows you to set the driver state when the Identity Manager server is started.

Auto start: The driver starts every time the Identity Manager server is started.

Manual: The driver does not start when the Identity Manager server is started. The driver must be started through Designer or Identity Console.

Disabled: The driver has a cache file that stores all of the events. When the driver is set to Disabled, this file is deleted and no new events are stored in the file until the driver state is changed to Manual or Auto Start.

Driver Parameters

The Driver Parameters section lets you configure the driver-specific parameters. When you change driver parameters, you tune driver behavior to align with your network environment.

The parameters are presented by category:

- "Driver Settings" on page 95
- "Subscriber Settings" on page 95
- "Resources" on page 99
- "Publisher Options" on page 100

Driver Settings

Custom Java Extensions: Select **Show** if you have developed custom Java classes to extend the driver shim's functionality. Otherwise, select **Hide**.

- **Document Handling:** Select **Implemented** if you have developed a custom Java class to process data as XML documents. Otherwise, select **None**.
  - **Class:** Specify the class by using a complete package identifier. For example, `com.novell.DocumentModifier`.
  - **Init Parameter:** Specify the parameter to pass to the init() method of the specified class. The init method is responsible for parsing the information contained in this string. Leave this field blank if the configuration string is not required for the class.
- **Schema:** Select **Implemented** if you have developed a custom Java class to provide the application schema to the driver and specify the **Class** and **Init Parameter** values. Otherwise, select **None**.

For more information, see Appendix B, "Using Java Extensions," on page 105.

Subscriber Settings

Authentication Method: Select the method for authentication with the RESTful service. The available options are:

- **Anonymous:** The user name and password is not required in Anonymous authentication method.
- **Basic:** The driver uses the specified ID and password for authentication when processing the requests.
- **OAuth2.0:** The driver uses the specified access token URL, ID and password for authentication when processing the request.

If **Anonymous** is selected, fill in the following parameters:

Parameters	**Description**
Authorization Header Fields	Click the + icon to create authorization header fields. • **Header Name:** If the remote server requires an authentication ID, specify the ID in the field. Otherwise, leave the field empty. • **Header Value:** Specify the authentication password for the remote server if you specified an header name. Otherwise, leave the field empty.
Truststore file	Specify the name and path of the keystore file containing the trusted certificates used when the remote server is configured to provide server authentication. For example, `C:\security\truststore`. Leave this field empty when server authentication is not used.

Parameters	Description
Set mutual authentication parameter	Specify **Show** to set mutual authentication information. Specify **Hide** to not use mutual authentication. - **Keystore file:** Specify the path and the name of the keystore file that contains the trusted certificates for the remote server to provide mutual authentication. For example, `C:\security\keystore`. Leave this field blank when mutual authentication is not used. **NOTE:** From 1.1.2.0400 release, the value for Keystore type must be pkcs12. - **Keystore password:** Specify the password for the keystore file. Leave this field blank when mutual authentication is not used.
Http Connection Timeout	Specify the HTTP connection timeout value. The driver waits for the time specified and terminates the HTTP connection. The timeout value must be greater than zero.
Proxy host and port	Specify the host address and the host port when a proxy host and port are used. For example: 192.10.1.3:18180. Or, if a proxy host and port are not used, leave this field empty.
HTTP errors to retry	Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces. For example, 307 408 503 504.
Base URL for REST Resources	Specify the common part of the REST resource URL. This is the part of the URL remaining after excluding the URL extension of the resource. For example, `http://ipaddress:port/`.

If **Basic** is selected, fill in the following parameters:

Parameters	Description
Authentication ID	Specify the authentication ID used for basic authorization on the HTTP header.
Authentication Password	Specify the authentication password used for basic authorization on the HTTP header.

Parameters	**Description**
Authorization Header Fields	Click the ✚ icon to create authorization header fields. ◆ **Header Name:** If the remote server requires an authentication ID, specify the ID in the field. Otherwise, leave the field empty. ◆ **Header Value:** Specify the authentication password for the remote server if you specified an header name. Otherwise, leave the field empty.
Truststore file	Specify the name and path of the keystore file containing the trusted certificates used when the remote server is configured to provide server authentication. For example, `C:\security\truststore`. Leave this field empty when server authentication is not used.
Set mutual authentication parameters	Specify **Show** to set mutual authentication information. Specify **Hide** to not use mutual authentication. ◆ **Keystore file:** Specify the path and the name of the keystore file that contains the trusted certificates for the remote server to provide mutual authentication. For example, `C:\security\keystore`. Leave this field blank when mutual authentication is not used. ◆ **Keystore password:** Specify the password for the keystore file. Leave this field blank when mutual authentication is not used.
Http Connection Timeout	Specify the HTTP connection timeout value. The driver waits for the time specified and terminates the HTTP connection. The timeout value must be greater than zero.
Proxy host and port	Specify the host address and the host port when a proxy host and port are used. For example: 192.10.1.3:18180. Or, if a proxy host and port are not used, leave this field empty.
HTTP errors to retry	Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces. For example, 307 408 503 504.
Base URL for REST Resources	Specify the common part of the REST resource URL. This is the part of the URL remaining after excluding the URL extension of the resource. For example, `http://ipaddress:port/`.

If **OAuth2.0** is selected, fill in the following parameters:

Parameters	Description
Access Token URL	Specify the URL of the server used for requesting token access.
User Name	Specify the user name for authentication. This parameter is optional.
User Password	Specify the password for authentication. This parameters is optional
Authorization Query Options	Click the + icon to create authentication query options for OAuth2.0 authorization method. The supported OAuth authorization types for REST driver are Client Credentials and `Resource Owner Credentials`. You can create any one of these authorization types. ◆ **Query Name:** Specify the name of the query. For example, `grant_type`. You also can configure `client_id`, `client_secret`, and `resource` as query names. ◆ **Query Value:** Specify the value for the query. For example, `client_crendentials` or `password`.
Authorization Header Fields	Click the + icon to create authorization header fields. ◆ **Header Name:** If the remote server requires an authentication ID, specify the ID in the field. Otherwise, leave the field empty. ◆ **Header Value:** Specify the authentication password for the remote server if you specified an header name. Otherwise, leave the field empty.
Truststore file	Specify the name and path of the keystore file containing the trusted certificates used when the remote server is configured to provide server authentication. For example, `C:\security\truststore`. Leave this field empty when server authentication is not used.

Parameters	Description
Set mutual authentication parameters	Specify **Show** to set mutual authentication information. Specify **Hide** to not use mutual authentication. ◆ **Keystore file:** Specify the path and the name of the keystore file that contains the trusted certificates for the remote server to provide mutual authentication. For example, `C:\security\keystore`. Leave this field blank when mutual authentication is not used. ◆ **Keystore password:** Specify the password for the keystore file. Leave this field blank when mutual authentication is not used.
Http Connection Timeout	Specify the HTTP connection timeout value. The driver waits for the time specified and terminates the HTTP connection. The timeout value must be greater than zero.
Proxy host and port	Specify the host address and the host port when a proxy host and port are used. For example: 192.10.1.3:18180. Or, if a proxy host and port are not used, leave this field empty.
HTTP errors to retry	Specify the HTTP errors that must return a retry status. Error codes must be a list of integers separated by spaces. For example, 307 408 503 504.
Base URL for REST Resources	Specify the common part of the REST resource URL. This is the part of the URL remaining after excluding the URL extension of the resource. For example, `http://ipaddress:port/`.

Resources

Configure Resources to synchronize: Click the + icon to add a class name of the user resource present in application schema.

- **Schema name:** Specify the class name of the user resource in the application schema. For example, Users, Groups, and Entitlement.
- **Configure Handlers:** Select the appropriate customer handlers. The available options are **Default** and **Custom**.

  If you select **Custom**, fill in the following parameters:
- **Rest Handler Details:** Click the + icon to add rest custom handler information.
- **URL Extension:** Specify the relative URL extension where the resource is located. The driver shim appends this URL extension to the base URL. The URL extension also includes the necessary URL placeholder. A placeholder is defined as a variable embedded within the URL. The `driver-operation-data` element replaces this with the URL token element during data transformation.

For example, /Users/<version>. In this example, version is the placeholder and the driver replaces this with the URL token element in the `driver-operation-data` element.

```
<driver-operation-data class-name="User" command="add" method="put"
uri="https://172.16.0.0:XXXX/User/rest123">
        <request>
          <url-token version="1.0"/>
          <header content-type="application/json"/>
          <value>{"CN":[{"value":"rest6789"}],"Full
Name":[{"value":"rest6789 rest6789"}],"Given
Name":[{"value":"rest6789"}],","Surname":[{"value":"rest6789"}],"Login
Disabled":[{"value":"true"}]}
          </value>
        </request>
</driver-operation-data>
```

- **Operation:** Select the required operation for Identity Manager operation.
- **Method:** Select the HTTP method to use. The options are: GET, POST,PATCH,PUT, and DELETE.
- **Optional Header Fields:** Click the + icon to add optional header name and value.

Publisher Options

Publisher Settings: Specify the publisher settings. You can select either **Publish Mode** or **Poll Mode** as the publisher setting. If **Publish Mode** is selected, the driver pushes the events to the Identity Vault. In the Publish mode, the driver exposes the REST endpoints to receive the events. These events are then pushed to the Identity Vault. If **Poll Mode** is selected, the driver periodically pulls the data from the connected RESTful service.

In **Publish Mode** is selected, fill in the following parameters:

Parameters	Description
Listening IP address and port	Specify the IP address of the server where the REST driver is installed and the port number that this driver listens on.
Authentication Method	Select the authentication method as **Anonymous** or **Basic**. If **Basic** is selected, fill in the following parameters: - **Authentication ID:** Specify the Authentication ID of the remote server to validate incoming requests. If you imported a sample configuration file, this field contains the IP address and port that you specified in the wizard. - **Authentication Password:** Specify the authentication password of the remote server to validate incoming requests.

Parameters	Description
KMO Name	Specify the KMO name to be used in eDirectory. When the server is configured to accept HTTPS connections, this name becomes the KMO name in eDirectory. The KMO name is the name before the “-” (dash) in the RDN. Leave this field empty when a keystore file is used or when HTTPS connections are not used.
Keystore file	Specify the keystore name and path to the keystore file. This file is used when the server is configured to accept HTTPS connections.
Keystore password	Specify the keystore file password used with the keystore file specified above when this server is configured to accept HTTPS connections.
Server key alias	Specify a Server key alias when this server is configured to accept HTTPS connections. Leave this field empty when a KMO name is used or when HTTPS connections are not used.
Server key password	When this server is configured to accept HTTPS connections, this is the key alias password (not the keystore password). Leave this field empty when a KMO name is used or when HTTPS connections are not used.
Require mutual authentication	When using SSL, it is common to do only server authentication. However, if you want to force both client and server to present certificates during the handshake process, you should require mutual authentication.
Heartbeat interval in minutes	Specify the heartbeat interval in seconds. Leave this field empty to turn off the heartbeat.

If **Poll Mode** is selected, fill in the following parameters:

Parameters	Description
Configure Resource for poll	Click the + icon to add a class name of the user resource present in application schema. ◆ **Schema name:** Specify the class name of the user resource in the application schema. ◆ **Service Endpoint:** Specify the service end point of the connected RESTful service for the publisher polling. A generic example is `http://ip:port/schema`. For users: `http://172.16.0.0:port/User?search-attr=`. ◆ **Method:** Select the method. ◆ **Optional Header Fields:** Click the + icon to add optional header name and value.
Polling interval in minutes	Specify the polling interval in minutes. Default is one minute. **NOTE:** The Subscriber Base URL is mandatory for the driver authentication when using the poll mode.
Heartbeat interval in minutes	Specify the heartbeat interval in minutes. Leave this field empty to turn off the heartbeat.

If **Anonymous** is selected, the values you specified for authentication ID and authentication password are cleared.

ECMAScript

Displays an ordered list of ECMAScript resource files. The files contain extension functions for the driver that Identity Manager loads when the driver starts. You can add additional files, remove existing files, or change the order the files are executed.

Global Configuration

Displays an ordered list of Global Configuration objects. The objects contain extension GCV definitions for the driver that Identity Manager loads when the driver is started. You can add or remove the Global Configuration objects, and you can change the order in which the objects are executed.

Global Configuration Values

Global configuration values (GCVs) are values that can be used by the driver to control functionality. GCVs are defined on the driver or on the driver set. Driver set GCVs can be used by all drivers in the driver set. Driver GCVs can be used only by the driver on which they are defined.

The REST driver includes several predefined GCVs. You can also add your own if you discover you need additional ones as you implement policies in the driver.

To access the driver's GCVs in Identity Console:

1. Click the **IDM Administration** tile.
2. Select the driver set that contains the driver whose properties you want to edit.
3. Locate the driver icon, then click the driver icon to display the driver's properties page.
4. Click **Global Config Values** drop down to display the GCV page.

To access the driver's GCVs in Designer:

1. Open a project in the Modeler.
2. Right-click the driver icon or line, then select **Properties > Global Configuration Values.**

   or

   To add a GCV to the driver set, right-clickthe driver set icon , then click **Properties > GCVs**.

The global configuration values are organized as follows:

- "Password Synchronization" on page 103
- "Permission Collection and Reconciliation" on page 104

Password Synchronization

These GCVs enable password synchronization between the Identity Vault and the connected system.

In Designer, you must click the icon next to a GCV to edit it. This displays the Password Synchronization Options dialog box for a better view of the relationship between the different GCVs.

In Identity Console, to edit the Password management options go to **Configuration > Global Configuration Values**, and then change it in your Password synchronization policy tab.

For more information about how to use the Password Management GCVs, see "Configuring Password Flow" in the *NetIQ Identity Manager Password Management Guide*.

Application accepts passwords from Identity Manager: If True, allows passwords to flow from the Identity Manager data store to the connected system.

Identity Manager accepts passwords from application: If True, allows passwords to flow from the connected system to Identity Manager.

Publish passwords to NDS password: Use the password from the connected system to set the non-reversible NDS password in eDirectory.

Publish passwords to Distribution Password: Use the password from the connected system to set the NMAS Distribution Password used for Identity Manager password synchronization.

Require password policy validation before publishing passwords: If True, applies NMAS password policies during publish password operations. The password is not written to the data store if it does not comply.

Reset user's external system password to the Identity Manager password on failure: If True, on a publish Distribution Password failure, attempts to reset the password in the connected system by using the Distribution Password from the Identity Manager data store.

Notify the user of password synchronization failure via e-mail: If **True**, notifies the user by e-mail of any password synchronization failures.

Permission Collection and Reconciliation

If you installed the Permission Collection and Reconciliation package, Identity Console and Designer display the following options. For more information about permission reconciliation feature, see "Synchronizing Permission Changes from the Connected Systems "in the *NetIQ Identity Manager Driver Administration Guide*.

Enable Permissions Collection and Reconciliation: Set the value of this parameter to **true** for allowing permission collection and entitlement assignment. By default, the value is set to **false**, which allows the driver to override any other conditions to reconcile custom entitlements.

Enable Permissions Reconciliation for all Custom entitlements: If the value of this parameter is set to **No**, it allows you to select the custom entitlements for reconciling them. By default, it is set to **Yes**, which allows reconciling of all custom entitlements.

Click the **Add** ➕ icon add custom entitlements you want to selectively onboard and specify **Assignment Attribute Name** for them.

B Using Java Extensions

The functionality of the REST driver can be extended by using Java. You use an API defined by Java interfaces to create your own custom Java classes that have access to the data passing through the Subscriber and Publisher channels. These classes read and interpret the data, and, optionally, modify the data.

You can also configure Java extensions in the Java class that is available in the driver shim (`com.novell.nds.dirxml.driver.rest.RESTDriverShim`). The Java class consists of four empty functions. You need to enter the functions to perform operations as required. For more information, see Driver Development Kit in the Identity Manager Developer Documentation page.

This section contains the following information on using Java extensions:

- “Overview” on page 105
- “Creating and Configuring Java Extensions” on page 106

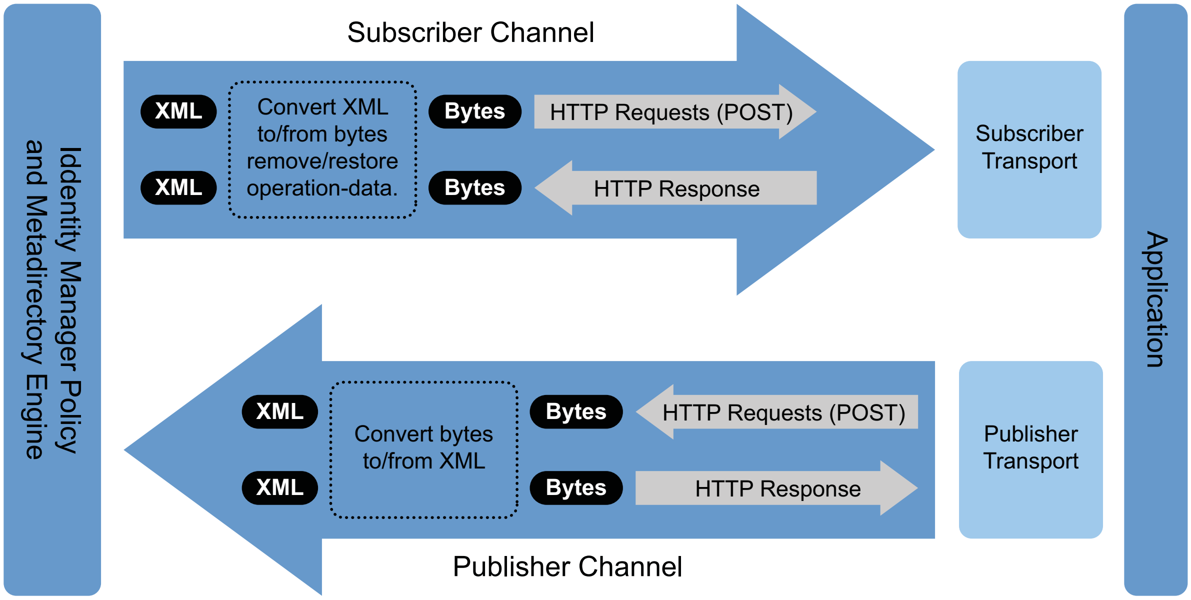
Overview

If the application you are using with the REST driver uses non-XML data that is not supported by the REST driver, you can create Java extensions to convert the non-XML data to the JSON format supported by the REST driver.

As illustrated in Figure B-1, there are five points where functionality can be extended:

- Two in the Subscriber channel
- Two in the Publisher channel
- One to report the application schema

Figure B-1 Using Java to Extend Functionality



The REST driver is designed to be flexible and extensible. For the Java programmer who wants to extend or modify the capabilities of the driver, there are programming interfaces that can be used for this purpose. These interfaces should be used only when you need to do transformations that cannot be done in policies or style sheets.

The Javadoc describes these interfaces.

There are two Java interfaces that can be used to extend or customize the driver behavior. They are DocumentModifiers and SchemaReporter.

DocumentModifiers is used to access and to modify the commands and events passing through the driver shim, if this is desired. DocumentModifiers gives you access to the data as XML DOM documents.

The other interface, SchemaReporter, can be used if you have a way of programatically determining the classes and attributes used by the remote Web service. The advantage to this is that creating schema mapping rules is easier if the schema can be dynamically determined.

Creating and Configuring Java Extensions

You should name your class by using any Java package and class name that is convenient for your environment and your organization.

For example, if you were writing your own class that implemented the DocumentModifiers interface, and you named your class *MyDocumentModifiers* within a package called `com.novell.idm`, then you would perform the following steps to compile, jar, and deploy your class:

1. Prepare your environment.

   Make sure you have a current Java Development Kit (JDK) installed on your computer. Visit the Java Web Site if you need to download one.

2 Gather your source code in the proper directory structure as defined by your package naming.

In the example given above, you would have a `com` directory that contained a `novell` directory that contained an `idm` directory. Within the `idm` directory, you would have a source file named `MyDocumentModifiers.java`.

3 Make sure you have the jar files you need to compile your class.

At a minimum, you need `RESTUtil.jar`. If you are using XML documents within your class, you also need `nxsl.jar`.

4 Put a copy of the required jar files in a convenient location like the root of your compile directory just outside the `com` directory, then access a system command prompt or shell prompt with that location as the current directory.

5 Compile your class by entering one of the following commands:

- **For Windows:** `javac -classpath RESTUtil.jar;nxsl.jar com\novell\idm\*.java`
- **For Linux or UNIX:** `javac -classpath RESTUtil.jar:nxsl.jar com/novell/idm/*.java`

6 Create a Java archive file containing your class by entering one of the following commands:

- **For Windows:** `jar cvf mydriverextensions.jar com\novell\idm\*.class`
- **For Linux:** `jar cvf mydriverextensions.jar com/novell/idm/*.class`

7 Place the jar file you created in Step 6 into the same directory that contains the `RESTShim.jar`.

In Windows, this is often `C:\Novell\NDS\lib`.

8 In Identity Console, edit the driver properties.

8a Next to Custom Java Extension, select **Show**.

8b Next to Document Handling, select **Implemented**.

8c Specify *com.novell.idm.MyDocumentModifiers* as the value for Class and any string as the value for Init Parameter.

The init parameter is the string that is passed to the init method of your class, so you can put any information here that you want to use during your class initialization.

9 Restart the driver.

You can now use your custom class.

C Trace Levels

The driver supports the following trace levels:

Table C-1 Supported Trace Levels

Level	Description
0	No debugging
1-3	Identity Manager messages. Higher trace levels provide more detail.
4	Previous levels along with Remote Loader, driver, driver shim, and driver connection messages, driver parameters, driver security, driver schema, request and response XML
5	Previous levels and driver shim debug level traces
6	Previous levels and all REST request
7	Previous levels and all REST responses from the connected system

For information about setting driver trace levels, see “Viewing Identity Manager Processes” in the *NetIQ Identity Manager Driver Administration Guide*.

D Supported JSON Format

The Identity Manager driver for REST queries the exposed RESTful endpoints and the returns the responses in JSON format.

The following is an example of the QUERY response in the supported JSON format.

```
{
    "totalResults": 1,
    "results": [
      {
            "src-dn": "\\SERVER-LINUX-TREE-45\\data\\users\\thomaswagner",
            "class-name": "User",
            "CN": [
                "thomaswagner"
            ],
            "Object Class": [
                "User",
                "Organizational Person",
                "Person",
                "ndsLoginProperties",
                "Top"
            ],
            "Password Allow Change": [
                "true"
            ],
            "Password Minimum Length": [
                "4"
            ],
            "Password Required": [
                "true"
            ],
            "Password Unique Required": [
                "false"
            ],
            "Public Key": [
                "AQAAAAQAAAAgAGAAAADWACc7sIe2QAUFVSU0FG"
            ],
            "Surname": [
                "thomaswagner"
            ],
            "Full Name": [
                "thomaswagner thomaswagner"
            ],
            "Revision": [
                "6"
            ],
            "Given Name": [
                "thomaswagner"
            ],
            "GUID": [
                "OTGey593Bkx1sDkxnsufdw=="
            ],
            "DirXML-Associations": [
                {
                    "nameSpace": "1",
                    "volume": "\\SERVERL-LINUX-TREE-
45\\system\\driverset1\\REST-DRIVER-PUB",
                    "path": "thomaswagner"
                },
                {
                    "nameSpace": "1",
                    "volume": "\\SERVERL-LINUX-TREE-
45\\system\\driverset1\\Data Collection Service Driver",
                    "path": "39319ECB-9F77-064c-75B0-39319ECB9F77"
                }
            ],
            "creatorsName": [
                "CN=linux-yal5,OU=servers,O=system"
            ],
            "modifiersName": [
                "CN=linux-yal5,OU=servers,O=system"
           ]
        }
    ]
}
```

The following is an example of the ADD request in the supported JSON format.

```
{
    "cn": "Sam2",
    "title": [
        "Sr Engineer",
        "Manager",
        "Mr. "
    ],
    "streetAddress": [
        {
            "component": "566666"
        },
        {
            "component": "area numero",
            "postal code": "566666"
        }
    ]
}
```

The following is an example of the MODIFY request in the supported JSON format.

```
{
    "cn": {
        "remove": "Sam1",
        "add": "Sam"
    },
    "title": {
        "add": [
            "Mr",
            "mr2"
        ]
    },
    "streetAddress": {
        "remove": [
            {
                "component1": "areanumero",
                "postalcode": "566666"
            }
        ],
        "add": [
            {
                "component2": " area numero ",
                "postalcode": "5555"
            }
        ]
    }
}
```

The following is an example of the GET response in the supported JSON format.

```
<nds dtdversion="4.0" ndsversion="8.x">
  <source>
    <product edition="Advanced" version="4.5.0.0">DirXML</product>
    <contact>NetIQ Corporation</contact>
  </source>
  <output>
    <status event-id="0" level="success"><driver-operation-data>
        <header Accept="application/json"/>
        <response>           <value>{"totalResults":2,"results":
[{"keyvalue1":{"NAME":"thomas","VALUE":29},
"keyvalue2":{"NAME":"wagnor","VALUE":30}}, {"search-attr":[{"Surname":
"Thomas"  }],"read-attr":["Surname","cn","Given Name"]}]}</value>
</response>
      </driver-operation-data>
    </status>
  </output>
</nds>
```