



UnboundID

UnboundID® Data Broker

Administration Guide

Version 5.2.0.1

UnboundID Corp
13809 Research Blvd., Suite 500
Austin, Texas 78750
Tel: +1 512.600.7700
Email: support@unboundid.com

Copyright

Copyright © 2016 UnboundID Corporation

All rights reserved.

This document constitutes an unpublished, copyrighted work and contains valuable trade secrets and other confidential information belonging to UnboundID Corporation. None of the material may be copied, duplicated, or disclosed to third parties without the express written permission of UnboundID Corporation.

This distribution may include materials developed by third parties. Third-party URLs are also referenced in this document. UnboundID is not responsible for the availability of third-party web sites mentioned in this document. UnboundID does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. UnboundID will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources. UnboundID and the UnboundID Logo are trademarks or registered trademarks of UnboundID Corp. in the United States and foreign countries. All other marks referenced are those of their respective owners.

Table of Contents

Copyright	i
Preface	ix
About UnboundID	ix
Audience	x
Documentation	x
Chapter 1: Introduction	1
Data Broker Overview	2
Data Broker Features	2
Data Broker Architecture	3
Data Broker Configuration Overview	5
Identity Provider Services	5
SCIM	6
Data Sources	6
Authorization and Policies	6
System	7
Logging, Monitoring, and Notifications	7
Sample Data Broker Configuration	8
Data Broker as both a Resource and Identity Provider Server	8
Data Broker as a Resource Server Only	9
Chapter 2: Installation	10
Installation Prerequisites	11
Supported Platforms	11
Set the File Descriptor Limit	11
Setting the Maximum User Processes	12
Installing the dstat Utility on SuSE Linux	12
Managing System Entropy	12
Installing the JDK	13
About Encryption Keys	13
User Store Overview	13
Installing the Data Store	13
Data Broker Installation Tools	15
Installing the Data Broker	15
Configuring the Data Broker	17

Logging into the Management Console	18
Installing Additional Data Brokers in a Topology	19
Server Folders and Files	20
Planning a Scripted Install	21
Scripted Installation Process	21
Installing the Data Broker with an Existing Truststore	22
Installing Sample Users	23
Run the Data Broker	24
Stop the Data Broker	24
Schedule a Server Shutdown	24
Run an In-Core Restart	25
Uninstalling the Data Broker	25
Using the Data Broker Sample Applications	26
The Profile Manager Application	26
The Sign-In Sample Application	31
User Account Registration and Recovery	35
Chapter 3: Data Access and Mapping	36
Data Components	37
Public Endpoints: UserInfo and SCIM	37
OpenID Connect Claims Map (UserInfo Map)	37
Store Adapters	37
Store Adapter Mappings	37
Data Stores	37
Store Adapter Overview	38
Primary and Secondary Store Adapters	38
Defining Correlation Attributes	38
Sample Configuration	39
SCIM Schemas	40
Store Adapter Mappings	40
SCIM Attribute Search Considerations	41
Maintaining Username Uniqueness	41
Defining SCIM Resource Types	42
Pass-through SCIM Resource Type	42

Mapping SCIM Resource Type Attributes	42
Creating a SCIM Resource Type	42
Editing Attribute and Sub-Attribute Properties	44
Editing Store Adapter Mappings	45
Defining OpenID Connect Claims	46
OpenID Connect Claims and Scopes	47
Complex Attribute Mapping	47
Creating an OpenID Connect Claims Map	47
OAuth2 Client-Specific SCIM Attributes	48
Chapter 4: Identity Provider Service and Scopes	50
OAuth2 Overview	51
OAuth2 Scopes	51
Authenticated Identity Scope	52
Resource Scope	53
Creating Scopes	55
Creating an Authenticated Identity OAuth2 Scope	56
Creating a Resource OAuth2 Scope	56
Identity Provider Configuration	56
Defining the Identity Provider Service	57
Creating an Identity Authenticator	60
Chapter 5: User Authentication	62
HTTP Authentication Schemes	63
OpenID Connect Request	63
OpenID Connect Response	63
The Data Broker as a Relying Party	64
Creating an Account through Identity Provider Login	65
Creating an External Identity Provider	65
Chapter 6: OAuth2 Clients and Token Access	68
OAuth2 Client Considerations	69
OAuth2 Authorization Grant Types	69
OAuth2 Authorization Response Types	70
Issuing Authorization Code Grant Requests	70
Example Redirection	71
Example Response	71
Example Request	71

Example Response	71
Example Request	72
Issuing Implicit Code Grant Requests	72
Example Redirection	72
Example Redirect Response	73
Example Request	73
Issuing Resource Owner Password Credentials Requests	73
Example Request	74
Example Response	74
Issuing Client Credentials Requests	75
Example Request	75
Example Response	75
Issuing ID Token Grant Requests	76
Adding an OAuth2 Client	78
The Data Broker Token Endpoint	79
Request	79
Response	79
Token Validation by the Data Broker	80
Token Revocation by the Data Broker	81
Obtaining a Refresh Token	81
Accepting External Access Tokens	82
The Data Broker Logout Endpoint	83
Request	83
Response	83
Chapter 7: Accessing Data	84
Data Broker Endpoints for OAuth2 Clients	85
The SCIM Endpoint	86
SCIM Examples	86
GET	86
GET (by User ID)	87
POST	88
UPDATE	89
DELETE	90

UserInfo Access Example	91
Request	91
Response	91
jQuery Example	92
Chapter 8: Configuring XACML Policies	93
XACML Policy Overview	94
Requesting an Access Token	94
Requesting Operations through SCIM or UserInfo	95
Policy Structure	96
Requesting JSON-Formatted Data	97
Using Obligations and Advice	97
Policies and Request Processing Per Endpoint	98
OAuth2 Endpoint Policy Requests	98
SCIM Resource Type Policy Evaluation	99
SCIM Sub-Resource Operation Policy Evaluation	102
UserInfo Endpoint Policy Evaluation	104
Policy Decision Point (PDP) Endpoint	104
Policy Engine Request Context	104
XACML Attribute Categories	104
Standard XACML Attribute Use	105
Custom XACML Function	106
Resource Properties	107
OAuth2 Client Properties	108
Scope Properties	108
HTTP Request Properties	110
SCIM Request Properties	110
Applicable Scopes	111
Session Properties	111
Access Token Properties	112
Policy Sections and Functions Described	112
Configuring the Policy Service	117
Policy Information Providers	117
PIP Evaluation Order	117
Creating XACML Policies	117
Creating a Policy Set	118

Testing Policies	118
Troubleshooting Policies with Traces	119
Troubleshooting Denied Access	119
Unsupported XACML Features	120
Chapter 9: Advanced Configuration	123
General Server Configuration	124
Available Configuration Tools	124
Using the dsconfig tool	125
Administrative Accounts	126
Using the Configuration API	128
Authentication and Authorization	129
Relationship Between the Configuration API and the dsconfig Tool	129
API Paths	137
Sorting and Filtering Configuration Objects	138
Updating Properties	139
Administrative Actions	140
Updating Servers and Server Groups	141
Configuration API Responses	141
Domain Name Service (DNS) Caching	142
IP Address Reverse Name Lookups	143
System Alarms, Alerts, and Gauges	143
Working with Logs and Log Publishers	146
Monitoring the Server	149
Server SDK Extensions	151
Data Broker Advanced Server Configuration	152
Configuring Third-Party Store Adapters	152
Example Third-Party Store Adapter	153
About Cross-Origin Resource Sharing Support	153
Managing Server Encryption Settings	156
Account Recovery Configuration in the Data Store	158
Configuring the Data Broker Templates	159
Supporting Multiple Content Types	161
Velocity Context Providers	162

Configuring HTTP Header Fields	162
Handling Specific HTTP Methods in Third-Party Providers	163
Velocity Tools Context Provider	164
Configuring the Broker Login and Consent Pages	164
Customizing the Data Broker Application Logo	166
Configuring Web Applications for Localization	167
Preserving Customized Files	168
Topology Configuration	168
Topology Master Requirements and Selection	168
Topology Components	169
Monitor Data For the Topology	170
Index	172

Preface

The UnboundID Data Broker Administration Guide contains concepts and procedures to configure an Identity Provider server, Resource server, or both. This includes defining Identity Provider settings, token requirements, XACML policies, OAuth2 clients, and the resources that can be requested. Management tasks and tools are also described.

About UnboundID

UnboundID Corp is a leading identity infrastructure solutions provider with proven experience in large-scale identity data environments. The Data Broker is part of the UnboundID Platform, which is an identity access and management platform—built specifically to handle the scale and real-time demands of millions of customers. The UnboundID Platform provides a unified view of customer data across all applications, channels, partners, and lines of business.

The UnboundID Platform provides the following:

- **Secure End-to-End Customer Data Privacy Solution** – A comprehensive identity platform with authorization and access controls to enforce privacy policies, control user consent, and manage resource flows. The platform protects data in all phases of its life cycle (create, read, update, delete, as well as static/unchanging and expiring).
- **Purpose-Built Platform** – Solutions to consolidate, secure, and deliver customer consent-given identity data. The platform provides unmatched security measures to protect sensitive identity data and maintain its visibility. The broad range of services include, policy management, cloud provisioning, federated authentication, data aggregation, and directory services.
- **Unmatched Performance across Scale and Breadth** – Support for the three pillars of performance-at-scale: users, response time, and throughput. The platform manages real-time data at large-scale consumer facing service providers.

- **Support for External APIs** – Standards-based solutions that can interface with various external APIs to access a broad range of services. APIs include XACML 3.0, SCIM, LDAP, OAuth2, and OpenID Connect.

Audience

This guide is intended for identity architects and administrators who are designing and implementing an identity infrastructure solution. Familiarity with system-, user-, and network-level security principles is assumed. Knowledge of directory services principles is recommended.

To use this guide effectively, readers should be familiar with the following subjects:

- REST web services and principles
- JSON or XML serialization formats
- XACML 3.0
- OAuth2 specification
- OAuth2 Bearer Token specification
- SCIM Schema 2.0
- OpenID Connect 1.0
- Apache Velocity Project and templates

Documentation

The Data Broker includes the following documents, available from the `index.html` page in the `docs` folder of the server.

- *UnboundID Data Broker Administration Guide (PDF)*
- *UnboundID Data Broker REST API Reference (HTML)*
- *UnboundID Data Broker Configuration Reference Guide (HTML)*
- *UnboundID Data Broker Command Line Reference (HTML)*
- *UnboundID Data Broker API Explorer*

Chapter 1: Introduction

Companies need to be able to monetize valuable user data, while balancing data privacy regulations. The Data Broker provides solutions to manage and monitor the authorization and authentication of user data access.

Topics include:

[Data Broker Overview](#)

[Data Broker Features](#)

[Data Broker Architecture](#)

[Data Broker Configuration Overview](#)

[Sample Data Broker Configuration](#)

Data Broker Overview

Most organizations today are working toward creating a unified customer profile. An essential part of creating that common identity profile is to centralize multiple, overlapping accounts and to define the logic for determining which applications should access data in a profile. The Data Broker enables managing large amounts of customer data while ensuring end-user privacy.

The Data Broker can act as a [Resource server](#), or both a [Resource server and Identity Provider server](#).

- As a Resource and Identity Provider server, the Data Broker provides authorization decisions for client applications, provisioning systems, API gateways and analytical tools in architectures involving personal, account, or sensitive identity data.
- As a Resource server, it provides restricted access to end users' information.

The Data Broker is designed to make authorization decisions based on XACML Policies and user consent. It is both the [policy decision point](#) and the [OAuth2 Identity Provider](#) for externalized authorization. Because the Data Broker centralizes the policy and consent functions, security rules are applied consistently across all applications. In addition, a common identity and single view of the customer can be configured by mapping account resources from multiple backend data stores to [SCIM Resource Types](#) defined in the Data Broker.

Data Broker Features

The Data Broker provides the following features for OAuth2 clients to securely access resources:

- **Support for multiple backend data stores.** The Data Broker supports multiple data stores, with native support for the UnboundID Data Store and extension points for others. Data stores serve as [user stores](#) to provide the resources that are requested by OAuth2 clients. OAuth2 clients can be written one time for access to the Data Broker and receive data from any type of infrastructure backend.
- **Standards-based authentication and authorization.** The Data Broker provides OAuth2 and OpenID Connect-compliant functionality for [authentication with the Data Broker](#) and [authorization to account resources](#). OpenID Connect provides the authentication layer on top of the OAuth2 protocol. It enables OAuth2 clients to verify the identity of a user based on the authentication performed by an Authorization Server, and obtain information about the user based on authorization flows and policy rules.
- **Authorization based on XACML policy and User Consent.** The Data Broker ensures that data is provided to authorized OAuth2 clients through the use of defined [OAuth2 Scopes](#) and [XACML policies](#). The XACML (eXtensible Access Control Markup Language) standard is used to define XML access control policies, and the processing model that

determines how to evaluate requests based on rules defined in the policies. Policies can be based on industry rules, corporate policy, or consent granted by customers.

- **SCIM Resource Types.** [SCIM Resource Types](#) determine what attributes can be accessed by an OAuth2 client through the Data Broker. The SCIM resource type defines the resource name, endpoint URL, schemas, and other metadata that indicate where a resource is managed and how it is composed.
- **Support for social login.** The Data Broker can act as a relying party, enabling users to log into OAuth2 clients and update or create Data Broker accounts with external identity provider accounts from Facebook, Google, or an OpenID Connect provider.
- **User interface samples and templates.** The [Profile Manager](#) and [Sample Sign-In](#) applications can be installed with the Data Broker to demonstrate how a client application makes requests for user data, how an end user can grant consent for the application to access that data, and how the Data Broker returns that data. [Data Broker templates](#) can be used for implementing custom user authentication and consent flows.
- **API Explorer.** The API Explorer is an interactive way to test data requests against various endpoints, and determine if authorization and XACML policy configuration is correct. The API Explorer works directly with the Data Broker so that configuration, testing, and updates can be done seamlessly. Access the API Explorer from the Documentation Index page, `<server-root>/docs/index.html`, or the server's HTTPS endpoint `https://<host>:<http-port>/explorer`.

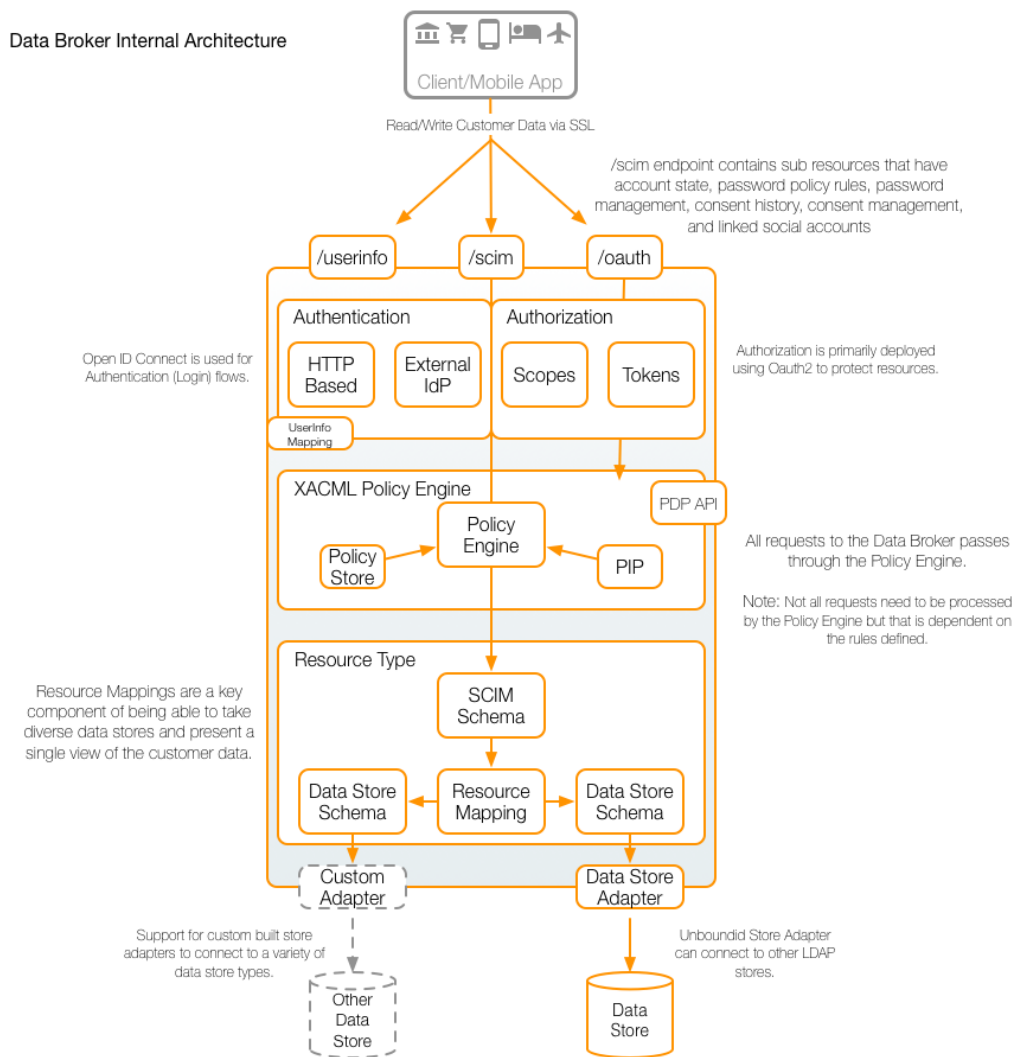
Data Broker Architecture

The Data Broker can act as both the Identity Provider and Resource server for OAuth2 clients requesting access to user data. Clients are granted authorization through an OAuth2 flow and receive access through OpenID Connect and SCIM endpoints. The Data Broker performs the following functions:

- Authorize an OAuth2 access token request, where the scopes requested represent resources that are served by the Data Broker's SCIM endpoint.
- Authorize an OAuth2 access token request, where the scopes requested represent resources that are served by an external Resource server.
- Authorize a resource (SCIM) request where the access token provided was generated by the Data Broker's Identity Provider Service.
- Authorize a resource (SCIM) request where the access token provided was generated by an external identity provider.

The following illustrates the Data Broker architecture and its components.

Chapter 1: Introduction



Planning a Data Broker deployment should start with determining its role as a Resource server. This includes defining what data can be accessed and updated from backend data stores, which can be configured as [User Stores](#) that supply or store user data. User Stores that have a [schema](#) defined can surface attributes and attribute properties. [SCIM Resource Types](#) are then defined to enable access to OAuth2 clients, and provide a unified view of identity data found in multiple data stores through [Store Adapter Mappings](#). [OAuth2 scopes](#) are created to define the resources that can be requested by an OAuth2 client and the actions that can be performed on those resources.

If using the Data Broker as an Identity Provider, the [Identity Provider Service](#) must be defined. This includes setting OAuth2 authorization token and OpenID Connect access token requirements, the default SCIM Resource Type, the [authentication scheme](#) and any [account registration or recovery](#) actions that can be performed. An [external identity provider](#) can also be configured to manage authentication.

[OAuth2 clients](#) that can request access to scopes are defined, including the [OAuth2 grant types](#) that can be used to access resources. Access token settings are inherited from the Identity Provider Service. Make sure that application development is done with consideration for the scopes that will be requested and how XACML policies will process these requests. See [Policies and Request Processing Per Endpoint](#).

[XACML policies](#) determine if a client can access requested scopes, based on the information provided with the request. Obligations within the policy can define conditions for access, such as requiring user's consent. XACML policies then determine the operations that can be performed on attributes within the requested scopes. Obligations can again define conditions for limiting access to certain attributes.

The Data Broker also tracks the consent that end users grant for access to their data. Consent can be managed by a requesting application or separate application through requested operations in OAuth2 scopes.

Data Broker Configuration Overview

Data Broker configuration defines all server services, policies, applications, resources, and the mapping of data from one or more backend data stores. Configuration can be done from the command line with the [dsconfig tool](#) or through the Management Console interface. All settings have associated help text in the interface and in the linked Configuration Guide. The Configuration Guide contains details and relationship specifics for all configuration objects and is available from the Management Console interface or from the `<server-root>/docs/index.html` page.

Identity Provider Services

Identity Provider Services contain the components and services that the Data Broker needs to process requests as an identity provider or through an external identity provider. If multiple Data Brokers are grouped in a topology, all configuration of these settings is mirrored across all servers in the topology. See [Topology Management](#) for more information. Identity Provider services include:

- **External Identity Providers** – Specifies the identity providers that can be used to log into the Data Broker, such as Google or Facebook.
- **Identity Authenticators** – Defines the authentication schemes that can be used to log into the Data Broker. This is required by the Identity Provider Service.
- **Identity Provider Service** – Defines the OAuth2 and OpenID Connect authorization and approval flow for self-service account management and access to Data Broker resources.
- **OAuth2 Client** – Specifies the OAuth2 clients that can request access to resources based on authorization and policy.
- **OpenID Connect Claims** – If using the `/userinfo` endpoint to access resources, claims are defined to determine the information that can be accessed.

SCIM

The SCIM protocol is an application-level, REST protocol for provisioning and managing identity data. The SCIM Schema provides a schema and extension for representing users and groups. Only those attributes defined in the SCIM Resource Type can be accessed through the Data Broker. Any changes to these settings are saved to all Data Brokers in a topology.

- **SCIM Resource Types** – Defines attribute mapping from a SCIM schema to native attributes found in data store entries, which provides a unified view of identity data found in multiple data stores. A pass-through SCIM Resource Type can also be created to allow the addition of new attributes that are not mapped to any in a data store. The SCIM schema defines the attributes that comprise a SCIM Resource Type. The SCIM Resource Type determines the attributes that can be accessed by a client application.
- **SCIM Schemas** – Specifies the SCIM 2.0 schemas for data that can be accessed from backend data stores. Schemas provide the basis for creating SCIM Resource Types.

Data Sources

Data sources are the servers that house the resources governed by the Data Broker.

- **External Servers** – Lists the LDAP data store instances that are configured with the Data Broker.
- **LDAP Health Checks** – Checks the status of external LDAP servers on a regular basis, and examines failures to determine if the server has become unavailable.
- **Load Balancing Algorithms** – Used to determine the appropriate LDAP external server to use to process a request. They may be used to provide improved availability and performance by distributing the workload across multiple backend servers.
- **Store Adapters** – Provides a data store interface to the Data Broker. Changes or additions to Store Adapters are saved to all Data Brokers in a topology. Third-party store adapters can be created with the UnboundID Server SDK.

Authorization and Policies

These settings define the rules for accessing resources through the Data Broker. Any changes to these settings are saved to all Data Brokers in a topology.

- **OAuth Scopes** – Specifies the data being requested with an OAuth2 authorization request from an OAuth2 client.
- **XACML Policies** – Specifies the rules for how requested resources can be shared with OAuth2 clients, based on the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0*. The Data Broker provides several default policies that can be used or modified.

- **XACML Policy Service** – Contains the properties that affect the overall operation of the Data Broker Policy Decision Point (PDP).

System

System settings define communication, connection, and the criteria for triggering alarms regarding the server's resources. Changes to these setting can be saved to the local server or saved to a group of servers. They are not mirrored across a topology. See [General Server Configuration](#) for more information.

- **Alarm Manager** – Defines the severity of alarms to be raised.
- **Connection Handlers** – Defines the settings for handling all interaction with the clients, including accepting connections, reading requests, and sending responses.
- **Gauges** – Specifies server performance thresholds and circumstances that merit the raising of an alarm.
- **Gauge Data Sources** – Defines the source of gauge data obtained from the server, including available memory and disk space.
- **Global Configuration** – Specifies the SMTP server, password policies, and LDAP request criteria configured for this server.
- **HTTP Servlet Cross Origin Policies** – Defines the configuration for handling Cross-Origin HTTP requests using the Cross Origin Resource Sharing (CORS) protocol. An instance of HTTP Servlet Cross Origin Policy can be associated with multiple HTTP Servlet Extensions.
- **HTTP Servlet Extensions** – Defines classes and initialization parameters used by a servlet invoked by an HTTP connection handler.
- **Locations** – Lists the locations in which servers that are accessed by the Data Broker reside.
- **Web Application Extensions** – Specifies the configuration settings for the Management Console and any other web applications that are configured to work with the Data Broker.

Logging, Monitoring, and Notifications

These settings define the notification criteria for system alerts, and the logging criteria for actions within the Data Broker. Changes to these setting can be saved to the local server or saved to a group of servers. They are not mirrored across a topology. See [General Server Configuration](#) for more information.

- **Alert Handlers** – Specifies the Alert Handlers used to notify administrators of problems or events that occur in the Data Broker.

- **Log Publishers** – Defines the distribution of log messages from different loggers to a destination.
- **Log Retention Policies** – Defines how long logs should be kept.
- **Log Rotation Policies** – Specifies when log files should be rotated.

Sample Data Broker Configuration

The following provides a reference sequence of tasks based on the role that the Data Broker will perform in an existing environment. These tasks can be performed from the [Data Broker Management Console](#) or with the [dsconfig tool](#). All components of the identity infrastructure should be identified before beginning system configuration.

Data Broker as both a Resource and Identity Provider Server

The following is a sample workflow for the Data Broker as both an Identity Provider and Resource server:

1. Determine how user data will be made available to OAuth2 clients. This includes determining the backend [user stores](#) that can be accessed, and how data across multiple stores will be correlated. A store adapter is installed with any LDAP data store, or third-party adapters can be created with the UnboundID Server SDK. A [SCIM Schema](#) can be used to surface attributes in the data store, and is needed if mapping attributes from multiple data stores to create a unified identity.
2. After SCIM Schemas are configured, resources from each configured user store are available for mapping. Configure [SCIM Resource Types](#) to make resources available to requesting OAuth2 clients. A SCIM Resource Type is required for the Identity Provider Service.
3. Configure the [Identity Provider Service](#) and the [authentication](#) used by the Data Broker. Settings enable the OpenID Connect and OAuth2 functionality, and self-service account flows, if needed.
4. Identify the [OAuth2 scopes](#) that can be accessed. OAuth2 scopes define the attributes that can be requested and the actions that can be performed. Scopes are required by OAuth2 clients when sending requests to the Data Broker.
5. Add the [OAuth2 clients](#) that can request access to data. The application client ID, client secret, scopes, and OAuth2 flows are defined with each client. This information will be needed by any client requesting data from the Data Broker.
6. Determine the XACML [policies](#) that will govern data access. Policies determine access based on the OAuth2 client making the request, the attributes requested, and the intended action to be taken on each attribute. Policies that are installed with the Data Broker are configured to use the OAuth2 scopes that are installed as well. Both can be

customized. If policies need to access decision-making information outside of the Data Broker configuration, a custom Policy Information Provider can be configured with the UnboundID SDK, or with the help of UnboundID Professional Services.

7. Policies determine what and how OAuth2 clients access resources. Make sure that policy rules work as expected by using [Policy Trace Filters](#) and other [Log Publishers](#) to verify that the requests to and responses from the Data Broker are as expected.
8. If using the `/userinfo` endpoint, data must be mapped from the Identity Provider SCIM Resource Type with [OpenID Connect Claims](#).
9. OAuth2 clients can be configured to surface an [external identity provider](#) (Facebook, Google, or OpenID Connect) for end users to log into the Data Broker.
10. If there is an UnboundID Metrics Engine installed, it can be configured to display system and consent metrics for the Data Broker. See the *UnboundID Metrics Engine Administration Guide* for information about configuring the Metrics Engine.

Data Broker as a Resource Server Only

If using the Data Broker as a Resource server only, resources will need to be created manually, and SCIM Resource Types configured. Configuration in this scenario will rely on an existing identity deployment and the type of authorization that the Data Broker is expected to provide. The following is a sample workflow for a Resource server:

1. Create the [OAuth2 Resource Scopes](#) that can be accessed by OAuth2 clients.
2. Register the [OAuth2 clients](#) that can request access to data. The application client ID, client secret, scopes, and OAuth2 flows are defined with the client.
3. Determine the XACML [policies](#) that will govern data access. Policies can base access decisions on the OAuth2 client making the request, the attributes requested, the environment information that is available, and the intended action to be taken on each attribute. Policies that are installed with the Data Broker are configured to use the OAuth2 scopes that are installed as well. Both can be customized.
4. Since the Data Broker is not the Identity Service Provider, a custom Policy Information Provider must be configured with the UnboundID SDK to validate access tokens from an external authorization server.
5. Policies determine what and how OAuth2 clients access resources. Make sure that policy rules work as expected by using [Policy Trace Filters](#) and other [Log Publishers](#) to verify that the requests to and responses from the Data Broker are as expected.
6. If using the `/userinfo` endpoint, map data with [OpenID Connect Claims](#).
7. OAuth2 clients can use an [external identity provider](#) (Facebook, Google, or OpenID Connect) accounts to access the Data Broker.

Chapter 2: Installation

The Data Broker installation requires few prerequisites, and can be deployed on virtualized and/or commodity hardware.

Topics include:

[Installation Prerequisites](#)

[About Encryption Keys](#)

[User Store Overview](#)

[Installing the Data Store](#)

[Installation Tools](#)

[Installing the Data Broker](#)

[Configuring the Data Broker](#)

[Logging into the Management Console](#)

[Installing Additional Data Brokers in a Topology](#)

[Server Folders and Files](#)

[Planning a Scripted Installation](#)

[Installing the Data Broker with an Existing Trust Store](#)

[Installing Sample Users](#)

[Running the Data Broker](#)

[Stopping the Data Broker](#)

[Uninstalling the Data Broker](#)

[Using the Sample Web Applications](#)

Installation Prerequisites

The following are required before installing the Data Broker:

- Java 7
- Minimum of 2 GB RAM
- UnboundID Data Store 5.2

Supported Platforms

The Data Broker is a pure Java application. It is intended to run within the Java Virtual Machine on any Java Standard Edition (SE) or Enterprise Edition (EE) certified platform. For the list of supported platforms and Java versions, access the UnboundID Customer Support Center portal or contact an authorized support provider.

Note

It is highly recommended that a Network Time Protocol (NTP) system be in place so that multi-server environments are synchronized and timestamps are accurate.

Set the File Descriptor Limit

The server allows for an unlimited number of connections by default, but is restricted by the file descriptor limit on the operating system. The file descriptor limit on the operating system can be increased with the following procedure.

Note

If the operating system relies on `systemd`, refer to the Linux operating system documentation for instructions on setting the file descriptor limit.

1. Display the current hard limit of the system. The hard limit is the maximum server limit that can be set without tuning the kernel parameters in the `proc` filesystem.

```
ulimit -aH
```

2. Edit the `/etc/sysctl.conf` file. If the `fs.file-max` property is defined in the file, make sure its value is set to at least 65535. If the line does not exist, add the following to the end of the file:

```
fs.file-max = 65535
```

3. Edit the `/etc/security/limits.conf` file. If the file has lines that set the soft and hard limits for the number of file descriptors, make sure the values are set to 65535. If the lines are not present, add the following lines to the end of the file (before `#End of file`). Insert a tab between the columns.

```
* soft nofile 65535
* hard nofile 65535
```

4. Reboot the server, and then use the `ulimit` command to verify that the file descriptor limit is set to 65535 with the following command:

```
ulimit -n
```

Once the operating system limit is set, the number of file descriptors that the server will use can be configured by either using a `NUM_FILE_DESCRIPTOR` environment variable, or by creating a `config/num-file-descriptors` file with a single line such as, `NUM_FILE_DESCRIPTOR=12345`. If these are not set, the default of 65535 is used. This is strictly optional if wanting to ensure that the server shuts down safely prior to reaching the file descriptor limit.

Setting the Maximum User Processes

Redhat Enterprise Linux Server/CentOS 6.x sets the default maximum number of user processes to 1024, which is lower than the setting on older distributions. This may cause JVM memory errors when running multiple servers on a machine because each Linux thread is counted as a user process. This is not an issue on Solaris and AIX platforms as individual threads are not counted as user processes.

At startup, the Data Broker attempts to raise this limit to 16,383 if the value reported by `ulimit` is less. If the value cannot be set, an error message is displayed. Explicitly set the limit in `/etc/security/limit.conf`. For example:

```
* soft nproc 100000
* hard nproc 100000
```

The 16,383 value can also be set in the `NUM_USER_PROCESSES` environment variable, or by setting the same variable in `config/num-user-processes`.

Installing the dstat Utility on SuSE Linux

The `dstat` utility is used by the `collect-support-data` tool to gather support data. It can be obtained from the OpenSuSE project website. Perform the following steps to install the `dstat` utility:

1. Log into the server as root.
2. Add the appropriate repository using the `zypper` tool.
3. Install the `dstat` utility:

```
$ zypper install dstat
```

Managing System Entropy

Entropy is used to calculate random data that is used by the system in cryptographic operations. Some environments with low entropy may have intermittent performance issues with SSL-based communication. This is more typical on virtual machines, but can occur in physical instances as well. Monitor the `kernel.random.entropy_avail` in `sysctl` value for best results.

Chapter 2: Installation

If necessary, update `$JAVA_HOME/jre/lib/security/java.security` to use `file:/dev/./urandom` for the `securerandom.source` property.

Installing the JDK

The Data Broker requires the Java 64-bit JDK. Even if Java is already installed, create a separate Java installation for use by Data Broker to ensure that updates to the system-wide Java installation do not inadvertently impact the Data Broker.

Solaris systems require the 32-bit and 64-bit versions. The 64-bit version of Java on Solaris relies on a number of files provided by the 32-bit version, so the latter should be installed first.

About Encryption Keys

Encryption setting definitions are used to protect Data Broker generated tokens and User Store metadata. All Data Broker instances must use the same set of definitions. Encryption setting definitions are managed using the `encryption-settings` tool.

If new encryption settings must be defined, the new definition must be exported using the `encryption-settings` tool and imported on all Data Broker instances. Only after the new definition is imported on all servers can the new definition be used for subsequent encryption operations.

See [Managing the Server Encryption Settings](#) for more information.

User Store Overview

During the Data Broker installation, at least one UnboundID Data Store is defined to serve as a user store, and to store user credentials. The user store is a repository of user data, such as names, email addresses, and preferences, as well as user-specific metadata needed by the Data Broker. For example, some user data may be stored in an LDAP directory server while other attributes may be stored in a relational database or a document database. [SCIM Resource Types](#) are defined to enable access to a user store's resources, and provide a consistent abstracted view of a user's profile that may cross multiple data stores.

Any LDAP data store added after the initial Data Broker installation must be configured with the `prepare-external-store` tool before it can be used as a user store. See [Data Broker Installation Tools](#).

Installing the Data Store

The Data Broker requires at least one installed UnboundID Data Store. This enables much of the account management and password recovery functionality. The Data Broker can be configured with multiple user stores.

Note

All sensitive data in the user store can be encrypted. When using the UnboundID Data Store

as the user store, server-level encryption can be enabled as described in the "Encrypting Sensitive Data" section in the *UnboundID Data Store Administration Guide*.

The following information is needed during the installation:

- Server hostname
- LDAPS port
- Root DN and password
- Base DN
- Location of user entries

All configuration settings can be later modified through the `dsconfig` tool.

Perform the following steps to install the Data Store:

1. Download the Data Store zip distribution, `UnboundID-DS-<version>.zip`.
2. Unzip the file in any location.

```
$ unzip UnboundID-DS-<version>.zip
```

3. Change to the top level UnboundID-DS folder.

```
$ cd UnboundID-DS
```

4. Run the `setup` command.

```
$ ./setup
```

5. Enter **yes** to agree to the license terms.
6. Enter the fully qualified host name or IP address of the local host, or press **Enter** to accept the default.
7. Create the initial root user DN for the Data Store, or accept the default, (`cn=Directory Manager`). This account has full access privileges.
8. Enter a password for this account, and confirm it.
9. To enable the Platform APIs over HTTPS, enter **yes**. These are the product's configuration APIs.
10. Enter the port to accept connections from HTTPS clients, or press **Enter** to accept the default. The default may be different depending on the account privileges of the user installing.
11. Enter the port to accept connections from LDAP clients, or press **Enter** to accept the default.
12. Type **yes** to enable LDAPS, or press **Enter** to accept the default (no). If enabling LDAPS, enter the port to accept connections, or press **Enter** to accept the default LDAPS port.

13. Type **yes** to enable StartTLS for encrypted communication, or press **Enter** to accept the default (no).
14. Select the certificate option for the server and provide the certificate location.
15. Specify the base DN for the Data Store repository, for example `dc=company,dc=com`.
16. Select an option to populate the database.
17. If this machine is dedicated to the Data Store, tune the JVM memory allocation to use the maximum amount of memory the **Aggressive** option). This ensures that communication with the Data Store is given the maximum amount of memory. Choose the best memory option for the system and press **Enter**.
18. Enter **yes** to configure the server on startup and load the backend into memory before accepting connections, or press **Enter** to accept the default (no).
19. To start the server after the configuration, press **Enter** for (yes).
20. Review the Setup Summary, and enter an option to accept the configuration, redo it, or cancel.

Data Broker Installation Tools

The Data Broker provides a number of tools to install and configure the system.

- The `setup` tool performs the initial tasks needed to start the Data Broker server, including configuring JVM runtime settings and assigning listener ports for the Broker's REST services and web applications.
- The `create-initial-broker-config` tool continues after `setup` and configures connectivity between the user store and the Data Broker. During the process, the `prepare-external-store` tool prepares each UnboundID Data Store to serve as a [user store](#) by the Data Broker. Configuration can be written to a file to use for additional installations.
- Once the configuration is done, the `dsconfig` tool and the Management Console enable more granular configuration.

Installing the Data Broker

To expedite the setup process, be prepared to enter the following information:

- An administrative account for the Data Broker.
- An available port for the Data Broker to accept HTTPS connections from REST API clients.
- An available port for the Management Console web application's communication.
- An available port to accept LDAP client connections.

- Information related to the server's connection security, including the location of a keystore containing the server certificate, the nickname of that server certificate, and the location of a truststore.

Perform the following steps for an interactive installation of the Data Broker:

1. Download the latest zip distribution of the UnboundID Data Broker software.
2. Unzip the file in any location.

```
$ unzip UnboundID-Broker-<version>.zip
```

3. Change to the top level UnboundID-Broker folder.
4. Run the `setup` command.

```
$ ./setup
```

5. Type **yes** to accept the terms of this license agreement.
6. The `setup` tool enables cloning a configuration by adding to an existing Data Broker topology. For an initial installation, press **Enter** to accept the default (no). When [adding additional Data Broker instances](#), an existing peer can be chosen to mirror configuration.
7. Enter the fully qualified host name or IP address of the machine that hosts the Data Broker, or press **Enter** to accept the default (local hostname).
8. Create the initial root user DN for the Data Broker. This account has full access privileges. To accept the default (`cn=Directory Manager`), press **Enter**.
9. Enter and confirm a password for this account.
10. Define a port for Data Broker Platform REST APIs to accept HTTPS connections, or press **Enter** to accept the default.
11. To enable the Management Console web application, press **Enter**. This is the web application used to manage the Data Broker server.
12. Enter the port for the Management Console to accept HTTPS connections, or press **Enter** to accept the default.
13. Enter the port to accept LDAP client connections, or press **Enter** to accept the default.
14. To enable LDAPS connections type **yes** and enter a port, or press **Enter** to accept the default (no).
15. To enable StartTLS connections over regular LDAP connection type **yes**, or press **Enter** to accept the default (no).
16. Enter the certificate option for this server. If needed, the server will generate self-signed certificates that should be replaced before the server is put into production.

17. If this machine is dedicated to the Data Broker, tune the JVM memory to use the maximum amount of memory (the **Aggressive** option). If this system supports other applications, choose an appropriate option.
18. Enter a location name for this server. The location is used for failover purposes if this server belongs to a server group.
19. Enter an instance name for this Data Broker, or press **Enter** to accept the default (<location> Broker 1). The name must be unique in a topology and cannot be changed once configured.
20. Press **Enter** (yes) to start the server when the configuration is applied.
21. Review the configuration options and press **Enter** to accept the default (set up the server).

The installation will continue with the `create-initial-broker-config` tool.

Configuring the Data Broker

The next set of steps in the setup process rely on the `create-initial-broker-config` tool. The `setup` tool will continue with the `create-initial-broker-config` tool to configure the Data Broker. Having the following in place will expedite the configuration:

- At least one UnboundID Data Store is installed. Have the host name and communication port available.
- Any additional Data Stores that act as user stores. Only UnboundID Data Stores can be configured with this tool. Other user stores must be configured outside of this process. Have the host names and communication ports available.
- Locations for this and any other Data Brokers for failover.

After the initial setup and configuration, run the `dsconfig` tool later to make configuration adjustments. Perform the following steps to configure the Data Broker:

1. Press **Enter** (yes) to continue with `create-initial-broker-config`. If for some reason the initial configuration cannot be completed in one session, manually restart `create-initial-broker-config` later.
2. Define the account used by the Data Broker to communicate with an external User Store, or press **Enter** to accept the default (`cn=Broker User,cn=Root` DNs,`cn=config`).
3. Enter and confirm the account password.
4. Specify the type of security that the Data Broker uses when communicating with all external store instances, or press **Enter** to accept the default (SSL).
5. Enter the `host:port` configured for the first Data Store. The connection is verified.

6. Select the location name for the Data Store (or user store server), or enter another location if not listed in the menu.
7. Confirm that the identified host should be prepared. If additional servers will be added as backups, select the **Yes, and all subsequent servers** option. This enables the [identification of another server](#) later in the configuration. The `prepare-external-store` tool can also be used to perform these tasks at a later time.
8. Enter the account and password needed to create the root user `cn=Broker` User, `cn=Root` DNs, `cn=config` account on the Data Store. This is the [root account](#) created in the initial setup, such as the default (`cn=Directory Manager`). The Data Broker sets up the DN and tests that it can access the account. This is also the account used to log into the Management Console.
9. To configure the initial user store, press **Enter** for (yes). The user store will be configured with a default Store Adapter and SCIM Resource Type, which will enable mapping of resources in the user store.
10. If there are additional data store locations, enter their `host:port`. If there are no additional servers to add, press **Enter** to continue.
11. Choose one of the predefined schemas (the standard user schema and optionally the reference application schema), or no schema.
12. Specify the base DN for locating user entries, such as `ou=people,dc=example,dc=com` and press **Enter**.
13. Review the configuration summary, and press **Enter** to accept the default (w) to write the configuration to a `dsconfig` batch file. The configuration is written to `<server-root>/resource/broker-cfg.dsconfig`. Certificate files are written to `external-server-certs.zip`.
14. Press **Enter** (yes) to confirm that the configuration should be applied to this Data Broker and written to the `broker-cfg.dsconfig` file.

This completes the initial configuration for the Data Broker. Run the `bin/status` tool to see that the Data Broker server is up and running.

Logging into the Management Console

After the Data Broker is installed, access the Management Console, `https://<host>:<port>/console/`, to verify the configuration. The root user DN or the common name of a root user DN is required to log into the Management Console. For example, if the DN created in [Configuring the Data Broker](#) was `cn=Directory Manager, directory manager` can be used to log into the Management Console.

Installing Additional Data Brokers in a Topology

A Data Broker instance can be cloned to serve as an additional server in a topology. Adding a server to an existing topology copies the original Data Broker's local configuration and links the two configurations. The configuration of Data Broker's cluster items and the topology settings are automatically mirrored across servers in a topology. See [Topology Overview](#) for details.

Note

When setting up a new Data Broker from an existing peer, the existing HTTP(S) connection handlers are not cloned. These connection handlers are created from scratch using default values of the new server and any specified port values.

1. Unpack the zip distribution in a folder different from the peer Data Broker.
2. Run the `./setup` command in the `<server-root>` directory of the new server.
3. Accept the licensing agreement.
4. Enter **yes** to add this server to an existing Data Broker topology.
5. Enter the host name of the peer Data Broker server from which the configuration will be copied.
6. Enter the port of the peer Data Broker.
7. Choose the security communication to use to connect to the peer Data Broker.
8. Enter the manager account DN and password for the peer Data Broker. The connection is verified.
9. Enter the fully-qualified host name or IP address of the local host (the new server).
10. Enter the HTTPS client connection port for the Data Broker, or press **Enter** to accept the default.
11. Select the option to install the Management Console application, if desired.
12. Enter the HTTPS connection port for the Management Console application, or press **Enter** to accept the default.
13. Enter the port on which the new Data Broker will accept connections from LDAP clients, or press **Enter** to accept the default.
14. Choose a certificate option for this Data Broker.
15. Choose the amount of memory to allocate to the JVM.
16. Choose the location for this server. The location of the peer is listed as an option, or a new location can be defined. Regardless, the new server will have its topology and cluster settings mirrored with its peer.
17. Enter a name for this server. The name cannot be changed after installation.
18. Press **Enter** to start the server after configuration.

19. Review the information for the configuration, and press **Enter** to set up the server with these parameters.
20. To write this configuration to a file, press **Enter** to accept the default (yes).

Server Folders and Files

After the distribution file is unzipped, the following folders and command-line utilities are available:

Directories/Files/Tools	Description
ldif	Stores any LDIF files that have been created or imported.
import-tmp	Stores temporary imported items.
classes	Stores any external classes for server extensions.
bak	Stores the physical backup files used with the backup command-line tool.
velocity	This directory is where custom velocity static files and templates reside. The server files are stored in <code>config/velocity</code> (and should not be modified directly). See Preserving Customized Files .
update.bat, and update	The update tool for UNIX/Linux systems and Windows systems. (Update is not supported for 5.2)
uninstall.bat, and uninstall	The uninstall tool for UNIX/Linux systems and Windows systems.
unboundid_logo.png	The image file for the UnboundID logo.
setup.bat, and setup	The setup tool for UNIX/Linux systems and Windows systems.
revert-update.bat, and revert-update	The revert-update tool for UNIX/Linux systems and Windows systems.
README	README file that describes the steps to set up and start the server.
License.txt	Licensing agreement for the product.
legal-notice	Legal notices for dependent software used with the product.
docs	Provides the release notes, Configuration Reference Guide (HTML), API Reference, and all other product documentation.
metrics	Stores the metrics that can be gathered for this server and surfaced in the UnboundID Metrics Engine.
bin	Stores UNIX/Linux-based command-line tools.
bat	Stores Windows-based command-line tools.
webapps	Stores the Management Console .war file and third-party licenses.
samples	Stores the sample application .zip files.
lib	Stores any scripts, jar files, and library files needed for the server and its extensions.
collector	Used by the server to make monitored statistics available to the Metrics Engine.
locks	Stores any lock files in the backends.

Directories/Files/Tools	Description
tmp	Stores temporary files.
resource	Stores the MIB files for SNMP and can include Idif files, make-Idif templates, schema files, dsconfig batch files, and other items for configuring or managing the server.
config	Stores the configuration files for the backends (admin, config) as well as the directories for messages, schema, tools, and updates.
logs	Stores log files.

Planning a Scripted Install

An interactive installation of an Data Broker uses the `setup` and `create-initial-broker` tools. This is the recommended installation method and should be used when possible. A scripted installation can be performed, for scenarios that require a custom configuration or automated deployment. The resulting `broker-cfg.dsconfig` batch file, created with the `create-initial-broker-config` tool, can then be used as a basis for scripted installations.

The following is performed by the `create-initial-broker-config` tool during an interactive installation:

External store preparation:

- For each UnboundID Data Store, the `prepare-external-store` tool is run. This updates the Data Store's schema, creates a privileged service account for use by the Data Broker with the DN `cn=Broker User,cn=Root DNs,cn=config`, and creates an administrative account.
- If the user store is comprised of LDAP directory servers, the `prepare-external-store` tool is run for every server that comprises the user store. This updates the directory server's schema, and creates a privileged service account for use by the Data Broker with the DN `cn=Broker User,cn=Root DNs,cn=config`.

Server configuration with `dsconfig`:

The `create-initial-broker-config` command has a `--dry-run` option that can be used to generate the `broker-cfg.dsconfig` file in non-interactive, or interactive mode, without applying the configuration to the local server.

Scripted Installation Process

The following is a sample of the commands that should be included in a scripted installation:

1. Set up and configure one or more Data Stores. See [Installing the Data Store](#).
2. Run the Data Broker `setup` command on the server that will host the Data Broker.

```

$./setup --cli --no-prompt --acceptLicense \
  --ldapPort <2389> --ldapsPort <2636> --httpsPort <8443> \
  --location <Austin> \
  --rootUserPassword <password> \

```

```
--useJavaTrustStore ~/tmp/keystores/truststore.jks \
--useJavaKeystore ~/tmp/keystores/broker1keystore.jks \
--trustStorePasswordFile ~/tmp/keystores/password.txt \
--keystorePasswordFile ~/tmp/keystores/password.txt \
--certNickname <server-cert>
```

3. Run `prepare-external-store` for each user store.

```
$ bin/prepare-external-store --no-prompt \
--port <1636> --useSSL \
--userStoreBaseDN "<o=Broker,dc=example,dc=com>"
--brokerBindPassword <password>
--bindDN "<cn=directory manager>"
--bindPassword <password>
```

4. Run the `create-initial-broker-config` tool.

```
$ bin/create-initial-broker-config --no-prompt \
--port <2636>
--bindDN "<cn=Directory Manager>"
--bindPassword <password>
--brokerBindPassword <password>
--externalServerConnectionSecurity useSSL
--userStoreBaseDN "<o=Broker,dc=example,dc=com>"
--userStore <d1.example.com:1636:Austin>
```

Installing the Data Broker with an Existing Truststore

By default, the `setup` command configures certificates and installs the keystore and truststore in the `config` directory (`config/keystore` and `config/truststore`). To use an existing keystore and truststore in a different path, run the `setup` tool, then run the `create-initial-broker-config` separately. The following procedures run `setup` from the command-line in non-interactive mode.

1. On the Data Broker, run `setup` non-interactively from the command line. In this example, the keystore and truststore passwords are the same. If the files are not already present in their paths, the command will fail.

```
$. ./setup --cli --no-prompt --acceptLicense \
--ldapPort 2389 --ldapsPort 2636 --httpsPort 8443 --rootUserPassword password \
--useJavaTrustStore ~/tmp/keystores/truststore.jks \
--useJavaKeystore ~/tmp/keystores/broker1keystore.jks \
--trustStorePasswordFile ~/tmp/keystores/password.txt \
--keystorePasswordFile ~/tmp/keystores/password.txt \
--certNickname server-cert
```

2. Run the `create-initial-broker-config` tool from the command line. Provide the paths to both the `--brokerTrustStorePath` and the `--trustStorePath` with their respective

password.

```
$. /bin/create-initial-broker-config \  
  --brokerTrustStorePath ~/tmp/keystores/truststore.jks \  
  --brokerTrustStorePasswordFile ~/tmp/keystores/password.txt
```

Installing Sample Users

The Data Broker provides a template to create a set of users (1000) that can be used by the [sample applications](#). The schema must be created from `<server-root>/resource/starter-schemas/reference-apps-make-ldif.template` and installed on the UnboundID Data Store. Once complete, a set of users (`user.0` through `user.999`) is available. Passwords for each are password.

Perform the following steps to modify the data store entries according to the directives in the LDIF file:

1. From the Data Store server root, stop the server.

```
$ <UnboundID-DS>/bin/stop-ds
```

2. From the Data Broker server root, create the users LDIF file from the template provided. A success message is displayed when complete.

```
$ <UnboundID-Broker>/bin/make-ldif \  
  --templateFile <UnboundID-Broker>/resource/starter-schemas/  
  reference-apps-make-ldif.template \  
  --ldifFile <UnboundID-DS>/ldif/reference-apps-user-entries.ldif
```

3. From the Data Store server root, import the users. A successful import message is displayed when complete.

```
$ <UnboundID-DS>/bin/import-ldif \  
  --ldifFile <UnboundID-DS>/ldif/reference-apps-user-entries.ldif \  
  --includeBranch dc=example,dc=com \  
  --rejectFile <UnboundID-DS>/ldif/reject.ldif
```

If sample data was loaded in the Data Store installation, add the following command to this step:

```
--overwriteExistingEntries
```

The [Profile Manager sample application](#) has an administrative view that enables editing a user's profile and consents for data access. This requires granting an entitlement to one of the users created. The sample schema template does not grant entitlements to any users. An entitlements file is available in `resources/starter-schemas/entitlements-ldap-modify.ldif` for adding the `admin` entitlement to `user.999`. If needed, it can be edited to add other entitlements or modify other users.

Note

This does not grant administrative rights to manage Data Broker configuration.

When installed, the Profile Manager setup creates a CSR resource scope, and it is the presence of that scope in the user's access token that the application uses to grant access to the administrative functionality.

Perform the following steps to add the `admin` entitlement to a user:

1. If necessary, start the Data Store.

```
$ UnboundID-DS>/bin/start-ds
```

2. From the Data Store server root, run the following command to add this file.

```
$ <UnboundID-DS>/bin/ldapmodify -p <ldap-port> \
--bindDN "cn=directory manager" \
--bindPassword password \
-f entitlements-ldap-modify.ldif
```

Run the Data Broker

To start the Data Broker, run the `bin/start-broker` tool on UNIX/Linux systems (the `bat` command is in the same folder for Windows systems).

To Run the Data Broker in the foreground:

1. Enter the `bin/start-broker` with the `--nodetach` option to launch the Data Broker as a foreground process.

```
$ bin/start-broker --nodetach
```

2. Stop the Data Broker by pressing CTRL-C in the terminal window where the server is running or run the `bin/stop-broker` command from another window.

Stop the Data Broker

The Data Broker provides a shutdown script, `bin/stop-broker`, to stop the server.

Schedule a Server Shutdown

The Data Broker enables scheduling a shutdown and sending a notification to the `server.out` log file. The server uses the UTC time format if the provided timestamp includes a trailing "Z," for example, 201304032300Z. The following example includes a `--stopReason` option that writes the reason for the shutdown to the logs:

```
$ bin/stop-broker --task \
--hostname <server1.example.com> \
--bindDN uid=admin,dc=example,dc=com \
--bindPassword <password> \
--stopReason "Scheduled offline maintenance" \
--start 201504032300Z
```

Run an In-Core Restart

Re-start the Data Broker using the `bin/stop-broker` command with the `--restart` or `-R` option. Running the command is equivalent to shutting down the server, exiting the JVM session, and then starting up again. Shutting down and restarting the JVM requires a re-priming of the JVM cache. To avoid destroying and re-creating the JVM, use an in-core restart, which can be issued over LDAP. The in-core restart will keep the same Java process and avoid any changes to the JVM options.

```
$ bin/stop-broker \  
--task \  
--restart \  
--hostname <server1.example.com> \  
--bindDN uid=admin,dc=example,dc=com \  
--bindPassword <password>
```

Uninstalling the Data Broker

The Data Broker provides an `uninstall` tool to remove the components from the system. If this instance is a member of a topology of Data Broker servers, the `uninstall` tool will remove it from the topology.

Note

If a Data Broker is a member of a topology, and the `uninstall` tool is not used to remove it (it was shutdown and deleted manually), it will not be removed from the topology registry. In this scenario, use the `bin/remove-defunct-server` tool to remove the instance from the topology.

Perform the following to uninstall the Data Broker:

1. From the server root directory, run the `uninstall` command.

```
$ ./uninstall
```

1. Select the option to remove all components or select the components to be removed.
2. To selected components, enter **yes** when prompted.

```
Remove Server Libraries and Administrative Tools? (yes / no) [yes]: yes  
Remove Log Files? (yes / no) [yes]: no  
Remove Configuration and Schema Files? (yes / no) [yes]: yes  
Remove Backup Files Contained in bak Directory? (yes / no) [yes]: no  
Remove LDIF Export Files Contained in ldif Directory? (yes / no) [yes]: no  
  
The files will be permanently deleted, are you sure you want to continue? (yes / no)  
[yes]:
```

3. Manually delete any remaining files or directories.

Using the Data Broker Sample Applications

Two sample applications can be installed in addition to the Management Console application. These applications are designed to illustrate a client's view of how data can be requested and delivered from the Data Broker, or an administrative view of a user's profile. Sample applications are located in `<server-root>/samples`.

To use these applications, sample users must be installed in the configured UnboundID Data Store. See [Installing Sample Users](#) for details. Applications can be customized.

Note

Because the sample applications use the Data Broker server for authentication, if bookmarking the application pages, bookmark the sample application landing pages, not the login page. Navigating to the bookmarked login page will cause authentication errors.

The Profile Manager Application

The Data Broker can be installed with a sample application called the Profile Manager. The interface operates as a customer portal to enable:

- Viewing consent for third-party access to the end-user's resources (typically from a web site).
- Revoking consent from OAuth2 clients.
- Editing a user's profile.
- Changing an account password.
- Accessing user accounts through an administrative (customer service) view (determined [by admin entitlement](#)) and:
 - Managing users' profiles, preferences, and consent.
 - Changing users' passwords.
 - Viewing user profile details (as JSON).
- Account linking to external identity providers, such as Facebook and Google.
- Editing account preferences, such as retail interests.

The following are provided with the application in `<server-root>/samples/profile-manager.zip`:

- `README.txt` – Describes how to configure and deploy the application either on the Data Broker server or on an external server.
- `profile-manager.war` – The packaged web application that can be deployed on an external server.
- `setup.dsconfig` – The script to install the sample application on the Data Broker server.

Deploying the Profile Manager Application

To deploy the sample applications, perform the following steps:

1. In the `<server-root>/samples` directory, unzip the `profile-manager.zip` file. The file must be unzipped in `/samples/profile-manager` to avoid errors during installation.
2. Review the `README.txt` file for instructions on deploying the application on an external server. Perform a simple deployment to the local server with the following command:

```
$ bin/dsconfig --batch-file setup.dsconfig
```

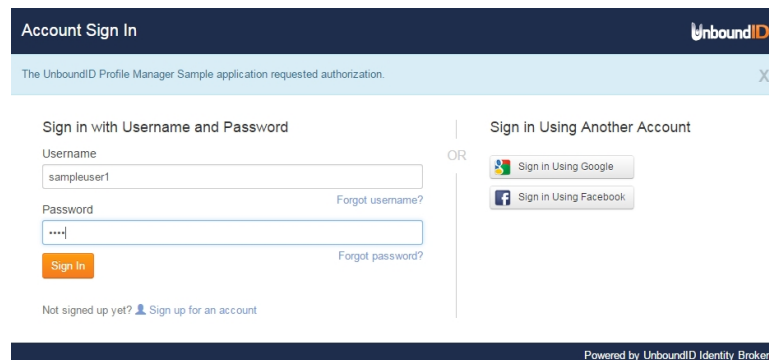
1. Launch the sample application in a browser with an address such as `https://<host:http-port>/samples/profile-manager`.

Profile Manager Application Pages

The following are the Profile Manager application's pages. Launch the application to view the template and login flows.

Sign In Page

This is the Data Broker login page, which can be configured from the Data Broker server. Sample users installed in the Data Store can be used to login, such as `user.0`, with password `password`.



An end user can log into the Data Broker. The account must exist in a data store that is configured to communicate with the Data Broker. If an external identity provider is configured, an icon for that provider is displayed on the login page. See [About the Data Broker as Relying Party](#) for information about the login and account creation flows.

If enabled, usernames can be recovered and passwords can be changed. Both require configuration on the Data Broker and the Data Store.

If enabled, a new user account can be created by clicking the **Sign up for an account** link.

User Search Page

If logging into the application as a user with the [admin entitlement](#), this page is displayed. End users will not see this page.

Enter a name, email address, or phone number to retrieve information for an end user. A new user account can also be created.

An existing user must reside in the backend user store that is configured for the Data Broker, and that user store must be mapped to a SCIM Resource Type in the Data Broker.

Account Registration

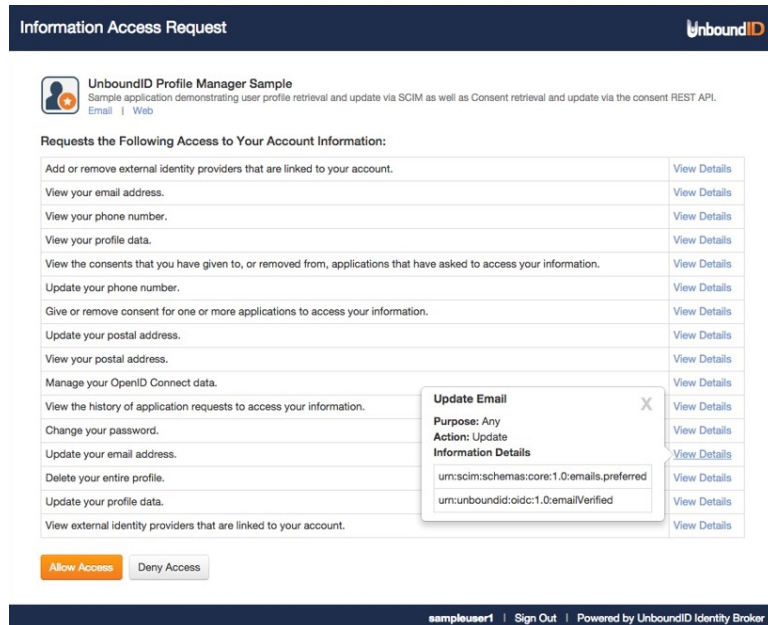
Account registration is configured in the Data Broker [Identity Provider Service](#). This is a Data Broker page. When configured, the following is displayed for account registration:

Enter the required information. The new account is added to the SCIM Resource Type configured for the [Identity Provider Service](#).

Information Access Request

If a user is logging into the application for the first time, the application can ask the user's consent to access account information. This is also a Data Broker page.

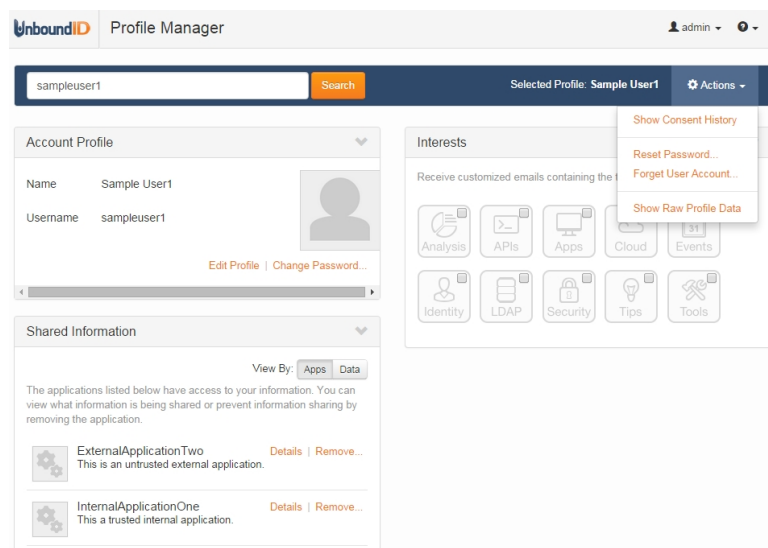
Chapter 2: Installation



The details for each information type show the specific attributes that are accessed.

Profile Results Page

The information that was retrieved or added for a user is displayed.



From this page, end users can perform the following:

- View and edit profile data.
- View consents granted to OAuth2 clients that request access to data.

- View and remove the OAuth2 clients that can access data.
- View and edit the types of information (Interests) that the user would like to see.

Username Recovery and Password Reset

If enabled on the server, the Data Broker can retrieve a username for an account or change a password. Both require an UnboundID Data Store configured as a user store. The recovery options are enabled in [Data Store Password Policies](#), and defined for the Data Broker in the [Identity Provider Service](#). On the Sign In page, click the **Forgot username?** link.

The user can enter information related to the account. The reCAPTCHA option is also configurable in the [Identity Provider Service](#). After the user clicks **Continue**, a verification code is delivered through a method selected on the Data Store.

The user enters the verification code. If the verification code is incorrect, and reCAPTCHA is enabled, the verification page will display a reCAPTCHA prompt for the user's next attempt.

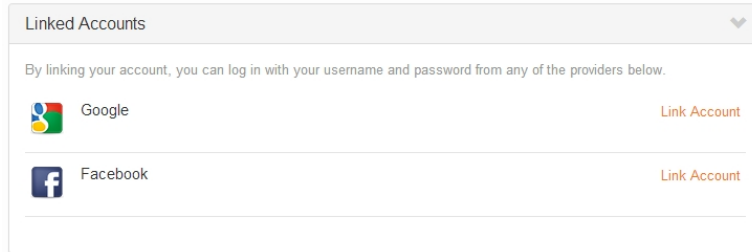
If verification succeeds and the account was found, the login page is displayed. If the account was not found, the verification will fail.

Note

No error is displayed to the user stating that the account was not found. This is to prevent phishing or any other type of exploitation that can be used by discovering which users are registered with this client. Text can be added to the server templates to help a user navigate to the next step.

Linked Accounts

If the Data Broker is configured to use an [external identity provider](#) as a login option, such as Google or Facebook, the identity provider and user accounts can be linked.



The Sign-In Sample Application

The Sign-In Sample application can be installed with the Data Broker and used as a model for an OAuth2 client using the OpenID Connect, including the `/userinfo` endpoint. The application demonstrates signing an end user into the Data Broker using the implicit grant flow, the Data Broker prompting the end user for consent to access resources, and the application retrieving the information that is configured in the [OpenID Connect Claims Map](#).

The following are provided with the Sign-In Sample application in `<server-root>/samples/sign-in.zip`:

- `README.txt` – Describes how to configure and deploy the application either on the Data Broker Server or on an external server.
- `sign-in.war` – The packaged web application that can be deployed on an external server. Included in this package is:
 - `ubid-broker-client.js` – A reusable script for the popup and redirect log in flows to the Data Broker server, and UserInfo claims retrieval. This script uses OpenID Connect and the [OAuth2 Implicit Grant authorization flow](#).
- `setup.dsconfig` – The script to install the sample application on the Data Broker server.

Deploying the Sample Application

Perform the following steps to deploy the sample application:

1. In the `<server-root>/samples` directory, unzip the `sign-in.zip` file. The file must be unzipped in `/samples/sign-in` to avoid errors during installation.
2. Review the `README.txt` file for instructions on deploying the application within the Data Broker server or on an external server. Perform a simple deployment to the local server with the following command:

```
$ bin/dsconfig --batch-file setup.dsconfig
```

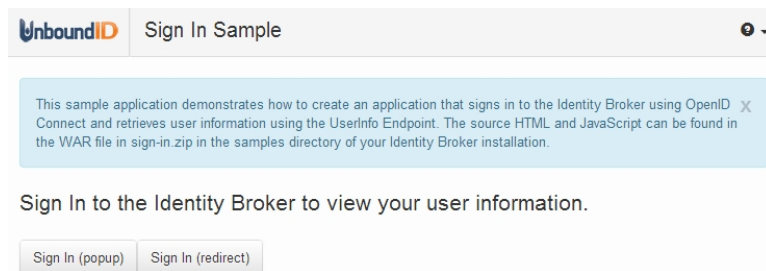
3. Launch the sample application in a browser with an address such as `https://<host:http-port>/samples/sign-in`.

Sign In Sample Application Pages

The following are the Sign In Sample application's pages. Launch the application to view and reuse the template and login flows.

Landing Page

When the application is launched, the landing page displays.

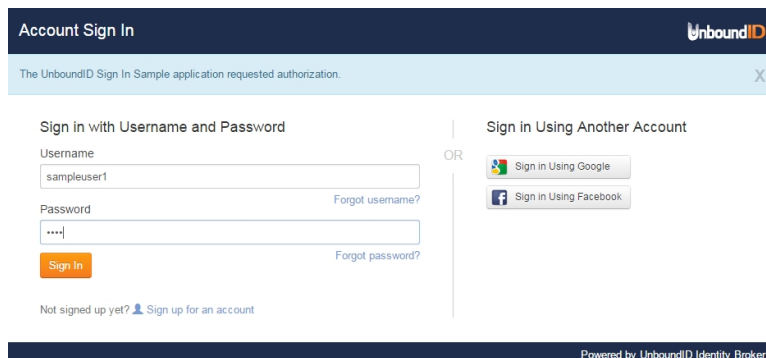


An end user can log in through a popup window, to maintain the client side state, or through a redirect, if a popup must be avoided. Both are provided in the sample.

Login Page

This is the Data Broker login page, which can be configured from the Data Broker server. The end user enters account credentials into the fields. The account must exist in an UnboundID Data Store that is configured to communicate with the Data Broker. If the OAuth2 client is configured to use an external identity provider to log in, an icon for that provider can be displayed on the page. See [About the Data Broker as Relying Party](#) for information about the login and account creation flows.

Sample users installed in the Data Store can be used to login, such as `user.0`, with password `password`.



Chapter 2: Installation

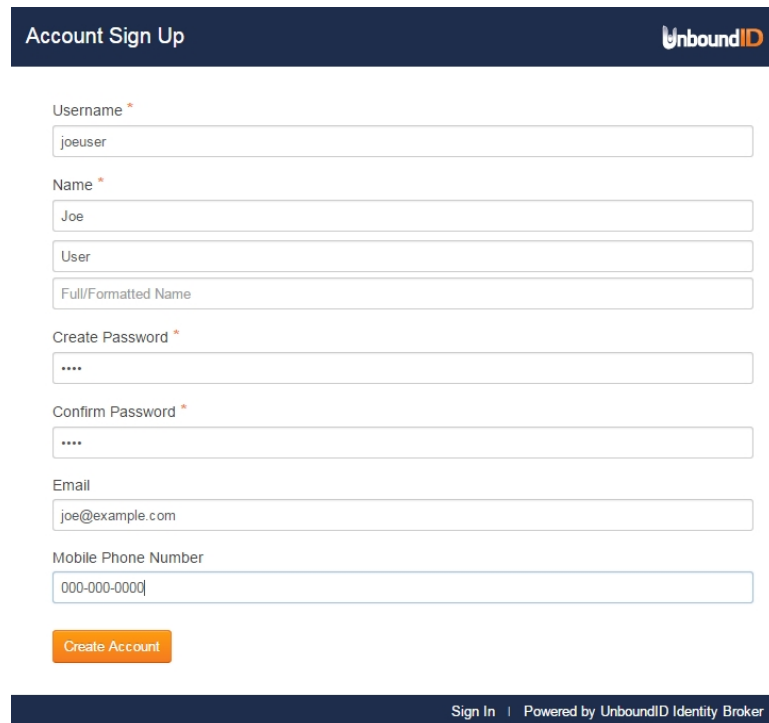
The application sends its client ID and a request to the Data Broker for the attributes in the requested scopes. If no scope is provided, the Data Broker will return the default values configured for the application.

If enabled, usernames and passwords can be recovered. See [Username Recovery and Password Reset](#). Both require configuration on the Data Broker server and the Data Store.

If enabled, a user account can be created by clicking the **Sign up for an account** link. This requires configuration on the Data Broker server.

Account Registration

If registering a new user account, the following is displayed:



The screenshot shows a web form titled "Account Sign Up" with the UnboundID logo in the top right corner. The form contains several input fields: "Username *" with the value "joeuser", "Name *" with the value "Joe", a "User" field, a "Full/Formatted Name" field, "Create Password *" with masked characters "****", "Confirm Password *" with masked characters "****", "Email" with the value "joe@example.com", and "Mobile Phone Number" with the value "000-000-0000". At the bottom of the form is an orange "Create Account" button. Below the form is a dark blue footer bar containing the text "Sign In | Powered by UnboundID Identity Broker".

Enter the required information. This is a Data Broker page. The new account is added to the default User SCIM schema and the Users SCIM Resource Type. This function is disabled by default. To enable it, enable the [Identity Provider Service](#) Register Enabled property.

Linked Accounts

If the application was configured to use an [external identity provider](#) as a login option, such as Google or Facebook, the identity provider and Data Broker accounts can be linked. This requires the configuration of specific scopes.

Confirm Consent Page

This is the Data Broker consent page, which can be configured from the Data Broker server. The client returns a request for end user consent.

Information Access Request

UnboundID

UnboundID Sign In Sample
Sample application demonstrating Identity Broker Sign In and UserInfo retrieval.

Requests the Following Access to Your Account Information:

View your profile data.	View Details
Manage your OpenID Connect data.	View Details

Allow Access

Deny Access

sampleuser1

Sign Out

Powered by UnboundID Identity Broker

Confirmation Page

If the end user clicks **Allow**, the approval page is displayed. The information that was retrieved from the OpenID Connect Claims Map is listed under User Information.

UnboundID

Sign In Sample

This sample application demonstrates how to create an application that signs in to the Identity Broker using OpenID Connect and retrieves user information using the UserInfo Endpoint. The source HTML and JavaScript can be found in the WAR file in sign-in.zip in the samples directory of your Identity Broker installation.

The authorize request was successful.

You are signed in.

Access Token

Aa7zDPoT9Th-LQLoUyUfWS79KafvAAAAAAAAAghtz82pwEz9F4bOv2QGWLtW6A1Q6DyeHA_0UTF99_D22SnEYG-_lpPo8ojjPSY6vn3ael8s0bAM3qlnTSrfnRapBE7RaJkCZ5dKNHNoTYjw

User Information (retrieved from the UserInfo endpoint with the Access Token)

sub
9f8a23-9dc49607-d8e6-4acc-889a-ba25ac423a92

updated_at
1429802415

name
Sample User1

family_name
User1

preferred_username
sampleuser1

given_name
Sample

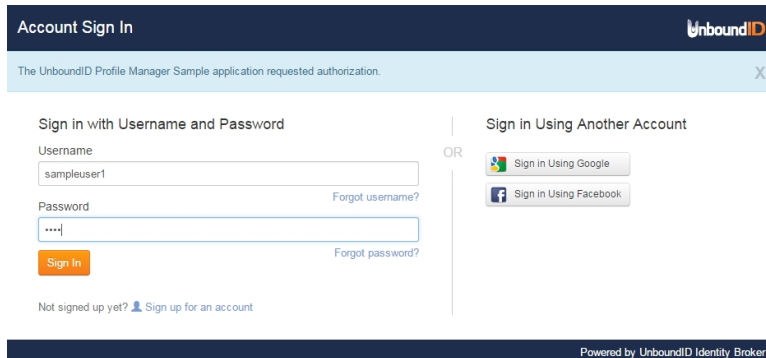
Sign Out

Sign Out

When an end user clicks **Sign Out**, the access token is invalidated but the user's consent remains intact for this client.

User Account Registration and Recovery

The Data Broker can register a new user, retrieve a username, or change a password for an account.



This requires the configuration of an UnboundID Data Store as the primary User Store. The templates for the login page and these functions are configured with Velocity templates. See [Configuring the Broker Login and Consent Pages](#).

Options for account recovery and new account registration are enabled by configuring the [Identity Provider Service](#) with the `dsconfig` tool or through the Management Console.

Note

Options for username recovery and password change are defined by the [Data Store password configuration](#).

If an account is not found, no error is displayed to the user. This is to prevent phishing or any other type of exploitation that can be used by discovering which users are registered with this application. Text can be added to the server templates to help a user navigate to the next step. If the end user does not receive a verification code, it may be a problem with the account information provided or the OTP Delivery Mechanism may be referencing an email or phone number that is not valid.

Chapter 3: Data Access and Mapping

Data stores provide the resources that can be accessed by OAuth2 clients. Attributes can be mapped from multiple data stores to create a unified identity in a SCIM Resource Type. The SCIM Resource Type is the component that makes resources available to OAuth2 clients.

Topics include:

[Data Components](#)

[Store Adapter Overview](#)

[Primary and Secondary Store Adapters](#)

[SCIM Schemas](#)

[Store Adapter Mappings](#)

[SCIM Attribute Search Considerations](#)

[Maintaining Username Uniqueness](#)

[Defining SCIM Resource Types](#)

[Defining OpenID Connect Claims](#)

[OAuth2 Client-Specific SCIM Attributes](#)

Data Components

When a data store is configured, a Store Adapter is installed to read and return native SCIM objects. Custom store adapters can be created for non-LDAP data stores with the UnboundID Server SDK. The attributes surfaced for each backend store are mapped in SCIM Resource Types to enable a unified view of a user profile, and to make them available to OAuth2 clients.

Public Endpoints: UserInfo and SCIM

The Data Broker, acting as a resource server, provides read access to user profile data through the UserInfo endpoint (`/userinfo`) and provides full read/write access through the SCIM Resource Type (`/scim/v2/me`). The access to these resources is subject to policy rules and restrictions.

OpenID Connect Claims Map (UserInfo Map)

A claims map maps OpenID Connect UserInfo claims to attributes defined in the SCIM Schema. Access to resources is read-only. Configure an OpenID Connect Claims map only if using the UserInfo endpoint.

Store Adapters

A store adapter connects the data coming into the Data Broker with an UnboundID Data Store or other external data store. For example, an LDAP Store Adapter manages the attribute mappings from an LDAP data store to a SCIM schema used for a corresponding SCIM Resource Type. The Data Broker provides an LDAP store adapter.

Store Adapter Mappings

A SCIM Resource Type enables attribute mappings between the native store adapter schema and the SCIM Schema. The Store Adapter mapping can contain additional information as to whether the native attribute is readable, writable, searchable, or authoritative. One must be authoritative. A SCIM Resource Type can map attributes from multiple data stores and determine which attributes are the authoritative resource for a user profile. See [Using SCIM Resource Type Attributes in Policy](#) for details about policy evaluation.

Data Stores

The data stores are the user repositories or data resources, which can be one or more UnboundID Data Stores, Proxy servers, or third-party directory servers. SCIM Resource Type mappings can be used to aggregate attributes from multiple data stores into a unified view.

When a Store Adapter is added to the Data Broker's server configuration, a correlation attribute must be defined for SCIM Resource Types that are backed by multiple store adapters. The correlation attribute defines an attribute for each Store Adapter that is used to uniquely

identify the same end user data across different store adapters. For example, if every data store stores a user's email address, and an email address can always be considered a primary key (that is, it is always unique per user), then each Store Adapter's email address attribute can be set as its correlation attribute.

Store Adapter Overview

A store adapter acts as an interface between the Data Broker's SCIM Resource Type layer and an external data store, such as an LDAP directory server, a relational database, or a REST service. A SCIM Resource Type can have one or more associated store adapters, each corresponding to a specific type of data store. When user data is retrieved or modified, the SCIM Resource Type calls the appropriate store adapter, which performs the actual operations against the data store, and passes results back up to the SCIM Resource Type layer.

The Data Broker provides a default store adapter that supports LDAP directory servers. Custom store adapters can be written using the UnboundID Server SDK. So that the translation between a store adapter and a SCIM Resource Type can be managed, store adapters expose user attributes as a SCIM schema. Attributes from the store adapter schema are mapped to attributes in the SCIM Resource Type schema.

Creating custom store adapters requires the UnboundID Server SDK. See [Server Extensions](#) for information.

Primary and Secondary Store Adapters

If the Data Broker is used to aggregate user attributes from multiple data stores, secondary store adapters can be configured. Store adapters contain the configuration that the Data Broker uses to interact directly with external data stores. Every data store providing a distinct set of user data must have a store adapter entry in the configuration.

"Primary store adapter" and "Secondary store adapter" designate how a SCIM Resource Type prioritizes user data lookups to multiple store adapters. The primary store adapter is always checked first when processing a request for a user resource, and then any secondary store adapters are checked. A user account effectively does not exist if a record does not exist for it on the primary store adapter. The primary store adapter should be used to store a user's core attributes, while a secondary store adapter can store additional attributes.

Defining Correlation Attributes

When handling a request for a particular user, the Data Broker needs a way to correlate an entry in the primary store adapter with any related entries in secondary store adapters. This is done by correlating the value of an attribute shared across the store adapters. This is configured using the secondary store adapter's `primary-correlation-attribute` and `secondary-correlation-attribute` properties. The correlation attribute should have a value that is unique for each user.

For example, user entries can be correlated across store adapters by email address:

Chapter 3: Data Access and Mapping

```
$ dsconfig create-secondary-store-adapter \  
  --type-name Users \  
  --adapter-name MarketingData \  
  --set store-adapter:DemographicsStoreAdapter \  
  --set primary-correlation-attribute:mail \  
  --set secondary-correlation-attribute:emailAddress
```

Sample Configuration

An environment may have two LDAP data stores with distinct sets of data. Set A may have user credentials and profile attributes, and is configured with the primary store adapter. Set B may have demographic data about these users, and is configured with the secondary store adapter. The following can be configured for this scenario:

1. Configure each server in Set A.

```
$ bin/dsconfig create-external-server \  
  --server-name profile-server \  
  --type unboundid-ds \  
  ...
```

2. Configure each server in Set B.

```
$ dsconfig create-external-server \  
  --server-name demographics-server \  
  --type unboundid-ds \  
  ...
```

3. Create LDAP load balancing algorithms.

```
$ dsconfig create-load-balancing-algorithm \  
  --algorithm-name "Profile Store LBA" \  
  --type failover \  
  --set enabled:true \  
  --set backend-server:profile-server
```

```
$ dsconfig create-load-balancing-algorithm \  
  --algorithm-name "Demographics Store LBA" \  
  --type failover \  
  --set enabled:true \  
  --set backend-server:demographics-server
```

4. Create Store Adapters.

```
$ dsconfig --adapter-name ProfileStoreAdapter \  
  --type ldap \  
  --set enabled:true \  
  --set "load-balancing-algorithm:Profile Store LBA" \  
  ...
```

```
$ dsconfig --adapter-name ProfileStoreAdapter \  
  --type ldap \  
  --set enabled:true \  
  --set "load-balancing-algorithm:Demographics Store LBA" \  
  ...
```

5. Designate the primary store adapter.

```
$ dsconfig create-scim-resource-type \
  --type-name Users \
  --type mapping \
  --set enabled:true \
  --set endpoint:Users \
  --set primary-store-adapter:ProfileStoreAdapter \
  --set core-schema:urn:example:schemas:Profile:1.0 \
  --set optional-schema-extension:urn:example:schemas:Demographics:1.0
```

6. Designate the secondary store adapter and correlation attributes.

```
$ dsconfig create-secondary-store-adapter \
  --type-name Users \
  --adapter-name MarketingData \
  --set store-adapter:DemographicsStoreAdapter \
  --set primary-correlation-attribute:mail \
  --set secondary-correlation-attribute:emailAddress
```

SCIM Schemas

Each SCIM Resource Type maps to one core SCIM Schema and optional extension schemas. SCIM schemas are used to define the resources that can be retrieved from a backend data store. Each SCIM Resource Type represents one type of resource, such as "user" or "account," and the schema defines the attributes of that resource.

Store Adapter Mappings

The Data Broker uses [Store Adapter Mappings](#) to determine which store adapter handles which attribute from the SCIM schema. The `secondary-store-adapter` property of a Store Adapter Mapping designates the store adapter to use.

The Data Broker can handle cases in which an attribute can be found on multiple store adapters. In these cases, one Store Adapter Mapping should be created for each combination of attribute and store adapter. One of these mappings must have the shared attribute set as `authoritative`. This designates the store adapter that will be the authoritative source when multiple possible values are found across a set of store adapters.

In the following example, the SCIM attribute `urn:unboundid:schemas:sample:profile:1.0:topicPreferences` is mapped to the LDAP attribute `ubidXTopicPreferenceJSON` from the Marketing Data store adapter:

```
$ bin/dsconfig create-store-adapter-mapping \
  --type-name Users \
  --mapping-name topicPreferences \
  --set secondary-store-adapter:DemographicsStoreAdapter \
  --set scim-resource-type-attribute:urn:example:schemas:Demographics:1.0:topicPreferences \
  --set store-adapter-attribute:ubidXTopicPreferenceJSON \
  --set authoritative:true
```

SCIM Attribute Search Considerations

In order to provide paging and sorting, the Data Broker holds an entire search result set in memory while it processes a SCIM search request. This is true for searches that do not request paging or sorting. The [SCIM Resource Type](#) `lookthrough-limit` property sets an upper bound for searches, so that clients do not exhaust the server resources. If the number of search results for a given request exceeds this value, an error is returned to the client indicating that the search matched too many results. A request that causes an unindexed search is also restricted to the size limit of the `lookthrough-limit` setting.

The Data Broker attempts to find a single store adapter that can process the provided search filter. The primary store adapter is checked first to see if it can process the search filter. If it cannot, the secondary store adapters are consulted in no particular order. The first store adapter capable of processing the search filter is chosen. The store adapter must be able to return a superset of possible matches for the filter. The attributes in the search filter must correspond to at least one searchable native attribute in the store adapter. If the SCIM Resource Type is a [Mapping SCIM Resource Type](#), the store adapter mapping for the search filter attribute must be marked as `searchable`.

If no store adapters can process the search, the Data Broker returns an error. For each candidate search result from a store adapter, the Data Broker assembles a complete SCIM resource by retrieving the native resource for every other store adapter using the store adapter correlation attributes (set when secondary store adapters are defined) and merging them together. Each resulting candidate SCIM resource is checked to see if it matches the provided search filter and is discarded if it does not match.

Maintaining Username Uniqueness

The Data Broker's default schema configuration uses "uid" as the RDN attribute of user DNs, which ensures that all uid values are unique for that branch of the DIT. In the default configuration, uid is recognized as a user's username. The following Data Broker functions rely on this:

- The `default-identity-autheticator` property of the [Identity Provider Service](#). This relies on a configuration object with a SCIM filter, which should be configured to only match zero or one resource.
- The Match Filter property of the default Username Password [Identity Authenticator](#).
- The [Store Adapter Mapping](#) for the `userName` attribute of the default starter schema.
- Username recovery and password reset. When made available to end users through the [Identity Provider Service](#), these features depend on `recover-username-search-filter` or `recover-password-search-filter` delivering a single result.

It may be the case that the attribute used for the username is also an RDN attribute in the data store. If every entry resides on the same branch, these attribute values will always be unique. Any configuration changes that do not maintain this structure must ensure that usernames are

unique. The UnboundID Data Store provides the attribute uniqueness plugin that can be used if configuration changes are required. See the *UnboundID Data Store Administration Guide*.

Defining SCIM Resource Types

SCIM Resource Types provide a unified view of resources between the Data Broker and one or more underlying data stores, and correspond to the SCIM 2.0 SCIM Resource Type. SCIM Resource Types determine what resources can be accessed from a data store. Each SCIM Resource Type represents one resource, such as "user" or "account" and the schema defines the attributes of that resource.

Note

When mapping attributes, data store attributes and SCIM Resource Type attributes must be of compatible types. For example, an attribute with an integer value must be mapped to another attribute with an integer value. An attribute with a string value can only be mapped to attributes with boolean, integer, or date-time if it can be parsed.

There are two types of SCIM Resource Types: Pass-through SCIM Resource Type and Mapping SCIM Resource Type. A Mapping SCIM Resource Type relies on a SCIM Schema, which is installed with the configuration of a user store on an UnboundID Data Store.

Pass-through SCIM Resource Type

This type of SCIM Resource Type simply exposes the primary store adapter's data as core attributes, while secondary store adapter's data are exposed as schema extensions. No schema needs to be defined at the SCIM Resource Type and all schema enforcement is at the responsibility of the store adapters. Since no schema is defined at the SCIM Resource Type, attribute mappings are not defined. If the configured store adapter exposes a schema, it will be enforced as the core or extension schemas for the SCIM Resource Type.

Mapping SCIM Resource Type Attributes

Attributes associated with a SCIM Resource Type are configured by specifying at least one core schema and one or more schema extensions. The core schema defines attributes that can appear at the root level of the SCIM resource exposed by the SCIM Resource Type. Schema extensions define attributes that are namespaced by the Schema's URI. Schema extensions can be optional or required. When processing client requests, the SCIM resource from the OAuth2 client is first checked against the schemas defined for the SCIM Resource Type (core or extension). The request is then mapped to a store adapter object, using the store adapter mappings, and then processed.

Creating a SCIM Resource Type

After user stores and Store Adapters are in place, SCIM Resource Types can be defined to provide a unified view of identity data found in multiple data stores. The SCIM Resource Type determines the attributes that can be accessed by an OAuth2 client.

The following is a sample command for creating a mapping SCIM Resource Type:

Chapter 3: Data Access and Mapping

```
$ bin/dsconfig create-scim-resource-type \  
  --type-name Users \  
  --type mapping \  
  --set "description:Users Resource Type" \  
  --set enabled:true \  
  --set endpoint:/Users \  
  --set primary-store-adapter:UserStoreAdapter \  
  --set core-schema:urn:unboundid:schemas:User:1.0 \  
  --set required-schema-extension:urn:unboundid:schemas:sample:profile:1.0
```

SCIM Resource Types can also be configured in the Management Console through **SCIM -> SCIM Resource Types**.

Creating a Mapping SCIM Resource Type

The following information is used to configure a Mapping SCIM Resource Type:

- A name for this SCIM Resource Type.
- An optional description for the SCIM Resource Type.
- The SCIM Resource Type's endpoint HTTP address, which will be relative to the `/scim/v2` base URL.
- A primary store adapter to persist the data for this SCIM Resource Type.
- The primary store adapter attribute to use as the value for the SCIM object ID. The object ID is a unique, immutable identifier for fetch, update, and delete operations on an object. The `entryUUID` attribute is the default for an LDAP store adapter.
- A look-through limit for the maximum number of resources that the SCIM Resource Type should scan when processing a search request. This prevents an OAuth2 client from taking too many of the server's resources for a single search.
- The core schema for the primary store adapter and any extension schemas.

Creating a Pass Through SCIM Resource Type

The following information is used to configure a Pass Through SCIM Resource Type:

- A name for this SCIM Resource Type.
- An optional description for the SCIM Resource Type.
- The SCIM Resource Type's endpoint HTTP address, which will be relative to the `/scim/v2` base URL.
- A primary store adapter to persist the data for this SCIM Resource Type.
- The primary store adapter attribute to use as the value for the SCIM object ID. The object ID is a unique, immutable identifier for fetch, update, and delete operations on an object. The `entryUUID` attribute is the default for an LDAP Store Adapter.

- A look-through limit for the maximum number of resources that the SCIM Resource Type should scan when processing a search request. This prevents an OAuth2 client from taking too many of the server's resources for a single search.

Editing Attribute and Sub-Attribute Properties

Attribute properties in the schema can be configured to change the actions that can be performed, and when an attribute is returned to a requesting OAuth2 client. If the attribute contains sub-attributes, those can be configured as well.

```
$ bin/dsconfig set-scim-attribute-prop \
--schema-name urn:unboundid:schemas:User:1.0 \
--attribute-name displayName \
--set "description:User's name."
--set required:true \
--set case-exact:true \
--set mutability:read-only
```

This can be configured in the Management Console by editing a schema in **SCIM -> SCIM Schemas**. Select a schema and edit any of the attributes listed. The following can be configured for an attribute or sub-attribute:

- An optional description of the attribute.
- The attribute type, which can be:
 - **string** - A sequence of zero or more Unicode characters encoded using UTF-8.
 - **boolean** - The literal `true` or `false`.
 - **datetime** - A date and time encoded as a valid `xsd:dateTime` (for example, 2008-01-23T04:56:22Z).
 - **decimal** - A real number with at least one digit to the left and right of the period.
 - **integer** - A decimal number with no fractional digits.
 - **binary** - Arbitrary binary data.
 - **reference** - A URI for a resource. A resource can be a SCIM resource, an external link to a resource (such as a photo), or an identifier such as a URN. The `reference-type` property must be specified for these attributes.
 - **complex** - A singular or multi-valued attribute whose value is a composition of one or more sub-attributes.
- Specify if the attribute is required.
- Specify if the attribute is case-sensitive.
- Specify if the attribute can have multiple values.
- Specify suggested canonical values that can be used (such as work and home).
- The circumstances under which the values of the attribute can be written (mutability). Values include:

- **read-only** - The attribute cannot be modified.
- **read-write** - The attribute can be updated and read.
- **immutable** - The attribute may have its initial value set, but cannot be modified after.
- **write-only** - The attribute can be updated but cannot be read.
- The circumstances under which the values of the attribute are returned in response to a request. Values include:
 - **by-default** - The attribute is returned by default in all SCIM responses where attribute values are returned.
 - **upon-request** - The attribute is returned in response to any PUT, POST, or PATCH operations if the attribute was specified by the client (for example, the attribute was modified).
 - **always** - The attribute is always returned.
 - **never** - The attribute is never returned.
- The SCIM Resource Types that can be referenced. This property is only applicable for attributes that are of type `reference`. Valid values are a defined SCIM Resource Type, `external` indicating the resource is an external resource (such as a photo), or `uri` indicating that the reference is to a service endpoint or an identifier (such as a schema urn).
- If the attribute is complex and has sub-attributes, they can be edited as well with these values.

Editing Store Adapter Mappings

Store adapters are designed to surface the schema of a backend data store. Store Adapter Mappings define a mapping between SCIM Resource Type attributes and store adapter attributes. When the Data Broker is installed with an UnboundID Data Store, the schema attributes are automatically mapped to a User SCIM Schema Resource Type.

Note

If the SCIM Resource Type attribute name changes, make sure that scopes and OpenID Connect Claims are updated to reflect the change.

The following is a sample command for editing a Store Adapter attribute mapping:

```
$ bin/dsconfig set-store-adapter-mapping-prop \  
--type-name Users \  
--mapping-name communicationOpts \  
--set store-adapter-attribute:ubidXCommunicationOptJSON \  
--set writable:false \  
--set searchable:true \  
--reset authoritative
```

Store Adapter Mappings can also be configured in the Management Console through **SCIM -> SCIM Resource Types**. Click **Actions -> Edit Store Adapter Mappings** for a SCIM Resource Type. The following is displayed:

Store Adapter Mappings - Users

Store Adapter Mappings define a mapping between SCIM Resource Type attributes and Store Adapter attributes.

Map New SCIM Resource Type Attribute... Bulk Edit... Search		
SCIM Resource Type Attribute ▲	UserStoreAdapter Mapping	Actions
accountVerified	ubidAccountVerified Read Write Search Auth	Remove
addresses	ubidPostalAddressJSON Read Write Search Auth	Remove
displayName	displayName Read Write Search Auth	Remove
emails	ubidEmailJSON Read Write Search Auth	Remove
entitlements	ubidEntitlement Read Write Search Auth	Remove
lastLogin	ds-pwp-last-login-time Read Write Search Auth	Remove

Individual attributes can be changed, or all can be edited by clicking **Bulk Edit**. For each attribute, the following can be configured:

- The store adapter attribute that is mapped to the SCIM Resource Type attribute.
- **Readable** – The SCIM Resource Type can read this attribute.
- **Writable** – The SCIM Resource Type can write to this attribute.
- **Searchable** – This specifies whether the attribute is efficiently searchable in the underlying data store. Indexed data store attributes determine what attributes (from the SCIM Resource Type Schema) can be used in a SCIM filter when performing a query. If an attribute is not indexed in the data store, it should not be marked as Searchable here.
- **Authoritative** – If there are multiple mappings for this attribute (from multiple data stores), one must be marked Authoritative.

Defining OpenID Connect Claims

A UserInfo endpoint is an OAuth2 protected resource that returns information about an authenticated end user. UserInfo Mapping enables mapping the Identity Service Provider's SCIM Resource Type attributes to claims returned from the UserInfo endpoint. The standard UserInfo data and claims are detailed in the OpenID Connect Authentication 1.0 Specification. Any custom claims can be defined and exposed at the UserInfo endpoint by adding (non-standard) entries in the UserInfo map.

OpenID Connect Claims and Scopes

For an OAuth2 client to successfully retrieve an OpenID Connect claim from the UserInfo endpoint, it must request and get consent to use a corresponding scope. Make sure that configured scopes contain the attributes that clients will request. Make sure that any changes to the SCIM schema or attribute mapping are also made in the scope configuration.

Complex Attribute Mapping

If an attribute is complex (such as `urn:scim:schemas:core:1.0:name`), the UserInfo endpoint returns a JSON object with property names matching the complex attribute's sub-attributes. For example, if `urn:scim:schemas:core:1.0:name` were mapped to a custom `name_object` OpenID Connect claim, the following would be returned for this claim:

```
"name_object":{"formatted":"Mort Kurio","familyName":"Kurio","givenName":"Mort"}
```

Sub-claims are mapped only if the OpenID Connect claim itself is correctly mapped to a SCIM Resource Type attribute.

Creating an OpenID Connect Claims Map

OpenID Connect Claims define a claim that can be exposed through the UserInfo endpoint, and its mapping to attribute(s) of the SCIM Resource Type that is defined for the [Identity Provider Service](#). Claims can be defined by name or the path, for example:

- `name` - Defines the `name` claim whose value is mapped from an attribute in the SCIM Resource Type that is defined for the Identity Provider Service.
- `name.last` - Defines the `name` claim whose value is a JSON object where the field `last` is mapped from an attribute in the SCIM Resource Type that is defined for the Identity Provider Service.
- `*` - All core or extension identity resource attributes are defined as claims with the same name and value.
- `urn:extension:*` - Maps all extension attributes identified by extension URN `urn:extension` in the SCIM Resource Type that is defined for the Identity Provider Service.
- `addresses[type eq "preferred"].postalCode` - Maps the `postalCode` sub-attribute of the address, where the sub-attribute type equals `preferred`.

The following is a sample command line for adding a claim:

```
$ bin/dsconfig create-openid-connect-claim \  
  --claim-name email_work \  
  --set 'identity-resource-attribute:emails[primary eq "true"].value'
```

Maps can also be edited and created in the Management Console under **Identity Provider -> OpenID Connect Claims**. The following information is needed to create a claims map:

- The name of the claim.
- The attribute name as represented in the SCIM Resource Type that is defined for the Identity Provider Service.

OAuth2 Client-Specific SCIM Attributes

Some environments may find it useful to designate a namespaced, schema-less portion of a SCIM user resource, in which an OAuth2 client can store its data. For example, a resource type could be configured such that an application may write any previously undefined attributes that are prefixed with `urn:customApp1`.

To enable this, the data store schema must first have a single-valued JSON attribute defined to hold application-specific attributes. For example, for an LDAP attribute called `customApp`:

```
customApp: { "urn:customApp1":{ "wine":["Napa Cabs","French Burgundy","Lodi Zinfandel"],
"age":"2000-2010" } }
```

This value should appear in the SCIM resource as follows:

```
'urn:customApp1' : {
  'wine' : [ 'Napa Cabs', 'French Burgundy', 'Lodi Zinfandel' ],
  'age' : '2000-2010'
}
```

The following is a command line sample of the steps needed to configure this type of functionality in the Data Broker, or this process can be done in the Management Console.

1. Create a store adapter mapping from "*" (SCIM) to "customApp" (LDAP). Using a wildcard SCIM attribute, client-specific SCIM attributes do not need to be defined in advance. To map only attributes from a single SCIM schema to an LDAP attribute, use a schema-specific SCIM wildcard such as `urn:myExtensionSchema:*`.

```
$ bin/dsconfig create-store-adapter-mapping \
--type-name "Users" \
--mapping-name "customAppWildcard" \
--set "scim-resource-type-attribute:*" \
--set store-adapter-attribute:customApp
```

2. Set the SCIM Resource Type's `schema-checking-option` property to `allow-undefined-attributes`.

```
$ bin/dsconfig set-scim-resource-type-prop \
--type-name "Users" \
--add schema-checking-option:allow-undefined-attributes
```

3. Define a wildcard scope that uses the client-specific namespace `urn:customApp1` as a prefix. Since the mapping is a wildcard, this prevents the client from reading or writing any user attribute, and client-specific attributes do not need to be defined in advance.

```
$ bin/dsconfig create-oauth2-scope \
--scope-name Wildcard-Scope \
--type authenticated-identity \
```

Chapter 3: Data Access and Mapping

```
--set "consent-prompt-text:Save application data to your account!" \  
--set "resource-attribute:urn:customApp1:*" \  
--set resource-operation:modify \  
--set resource-operation:retrieve
```

4. Create the OAuth2 client and assign the wildcard scope to it.

```
$ bin/dsconfig create-oauth2-client \  
--client-name "App1" \  
--set client-id:<App-ID> \  
--set client-secret:<secret> \  
--set grant-type:authorization-code \  
--set grant-type:implicit \  
--set scope:openid \  
--set scope:email \  
--set scope:Wildcard-Scope \  
--set redirect-url:https://company.com:<port>/client/
```

Chapter 4: Identity Provider Service and Scopes

Scopes define the attributes that an OAuth2 client can request, the name that is displayed to end users, the claims that can be accessed, and the actions that can be performed on each attribute. Scopes must be defined in the Data Broker server before a client can include them in requests. Scopes are also used to capture consent for the requested resources.

The Identity Provider Service defines the OAuth2 and OpenID Connect properties for access to the Data Broker. Self-service account registration, username retrieval, and password reset configuration is also defined here.

Topics include:

[OAuth2 Overview](#)

[OAuth2 Scopes](#)

[Creating Scopes](#)

[Identity Provider Settings](#)

OAuth2 Overview

The Data Broker, as an Identity Provider, uses the OAuth2 authorization framework, which enables clients to obtain access to protected resources by using tokens. The security and privacy of user information relies on the access requirements and consent flows configured for the [OAuth2 client](#).

OpenID Connect, built on the OAuth2 standard, is the identity layer that enables clients to authenticate end users without performing the authentication themselves. It also enables end-user identity data to be shared between interested parties with the end-users' consent. It provides two primary mechanisms for doing this:

- ID tokens. ID tokens are compact objects which identifies the user making the request and provide information about authentication events.
- The UserInfo endpoint. This is a bearer token-protected REST endpoint which provides attributes ("claims") about the identity of the access token owner.

The OAuth2 implementation, defined in the [Identity Provider Service](#), provides the necessary interfaces to define access requirements and develop an OAuth2 client. After the Data Broker is installed, the Identity Provider Service can be configured with the `dsconfig` tool or through the Management Console.

The encryption and decryption keys used to protect tokens and authorization codes are stored in the encryption settings database. See [Managing Server Encryption Settings](#) for information.

OAuth2 Scopes

When an OAuth2 client makes an authorization request using the standard OAuth2 endpoints, it specifies the level of access that it requires using scopes. Based on the application's configuration, the XACML policies that process the request, and consents granted by a user, the Data Broker will decide which scopes to return in an access token.

There are three scope types:

- Generic OAuth2 scope (used for external Resource servers).
- Authenticated Identity scope.
- Resource scope.

Consents can be captured for any scope requested during authorization by using the [prompt consent obligation](#) in a [XACML Policy](#).

A Generic OAuth2 scope includes the following properties, which are the base properties for the Authenticated Identity and Resource scopes.

Generic OAuth2 Scope Properties

Property	Description
Token Name	The scope name as presented in an OAuth2 request.
Type	The scope type, which is <code>oauth2</code> for generic scopes.
Description	A description of the scope for administrative use.
Consent Prompt Text	A description of the scope that will be presented in a consent dialog.
Tags	A list of Tags associated with this scope. Tags are arbitrary additional properties that can be examined by XACML policies.

Authenticated Identity Scope

This scope is granted for an authenticated end user. Once granted, the scope can be used to access the attributes of that authenticated identity. The attributes can be obtained through SCIM endpoints using the `/Me` authenticated subject alias as well as the URI of the SCIM resource, or obtained as OpenID Connect claims using the `/UserInfo` endpoint.

Properties in this scope include those in the standard OAuth2 scope and the following properties. At least one of the operation properties must have a value. Policy processing of requests that contain account, consent, or external identity provider operations is described in [SCIM Sub-Resource Operation Policy Evaluation](#).

Authenticated Identity Scope Properties

Property	Description
Type	The scope type, which is <code>authenticated-identity</code> for authenticated identity scopes.
Resource Operations	Operations can include <code>retrieve</code> (GET) or <code>modify</code> (PATCH or PUT) to endpoint <code>/scim/v2/<id></code> .
Account Operations	<p>Operations can include:</p> <ul style="list-style-type: none"> <code>reset-password</code> (PUT) to endpoint <code>/scim/v2/<id>/password</code> <code>retrieve-password-quality-requirements</code> (GET) from endpoint <code>/scim/v2/<id>/passwordQualityRequirements</code> <code>retrieve-account-state</code> (GET) from endpoint <code>/scim/v2/<id>/account</code> <code>replace-account-state</code> (PUT) to endpoint <code>/scim/v2/<id>/account</code>
Consent Operations	<p>Operations can include:</p> <ul style="list-style-type: none"> <code>retrieve-consent</code> (GET) from endpoint <code>/scim/v2/<id>/consents</code> or <code>/scim/v2/<id>/consents/<id></code>

Authenticated Identity Scope Properties	
Property	Description
	<ul style="list-style-type: none"> • <code>revoke-consent</code> (DELETE) from endpoint <code>/scim/v2/<id>/consents/<id></code> • <code>retrieve-consent-history</code> (GET) from endpoint <code>/scim/v2/<id>/consentHistory/<id></code>
	<p>Operations can include:</p> <ul style="list-style-type: none"> • <code>retrieve-external-identity</code> (GET) from endpoint <code>/scim/v2/<id>/externalIdentities</code> or <code>/scim/v2/<id>/externalIdentities/<id></code> <p>This will expose access tokens from the identity provider.</p> <ul style="list-style-type: none"> • <code>unlink-external-identity</code> (DELETE) from endpoint <code>/scim/v2/<id>/externalIdentities/<id></code>
External Identity Operations	
	<p>A list of one or more SCIM attributes of the authenticated identity for which this scope allows access. The type of access is determined by the operation properties <code>retrieve</code>, <code>replace</code>, and <code>modify</code>. A wildcard value of <code>*</code> can be used for all attributes. A schema-specific wildcard value of the form <code>urn:<schemaName>:*</code> can be used to represent all attributes of a single schema namespace. Access to attributes allowed per operation is the union of all <code>resourceAttributes</code> allowed in the scope.</p>
Resource Attributes	

Resource Scope

An OAuth2 scope that allows an OAuth2 client bearing a granted token to access resources of a specified SCIM Resource Type. It defines the SCIM operations (search, create, retrieve, update, and delete) that can be performed by the client, and the attributes that can be retrieved or updated. A Resource scope is similar to an Authenticated Identity scope, but potentially allows access (subject to XACML policy) to all resources of a specified SCIM Resource Type.

Properties in this scope include those in the Authenticated Identity scope and the following properties. At least one of the operations must have a value. Policy processing of requests that contain account, consent, or external identity provider operations is described in [SCIM Sub-Resource Operation Policy Evaluation](#).

Resource Scope Properties	
Property	Description
Type	The scope type, which is <code>resource</code> for resource scopes.
Scim Resource Type	The SCIM Resource Type that can be accessed with this scope.
	<p>Operations can include:</p> <ul style="list-style-type: none"> • <code>create</code> (POST) to endpoint <code>/scim/v2</code> • <code>search</code> (GET) from endpoint <code>/scim/v2</code>
Resource Operations	

Resource Scope Properties	
Property	Description
	<ul style="list-style-type: none"> • retrieve (GET) from endpoint <code>/scim/v2/<id></code> • replace (PUT) to endpoint <code>/scim/v2/<id></code> • modify (PATCH) to endpoint <code>/scim/v2/<id></code> • delete (DELETE) from endpoint <code>/scim/v2/<id></code>
	Only allowed if the SCIM Resource Type is User. Operations can include: <ul style="list-style-type: none"> • resetPassword (PUT) to endpoint <code>/scim/v2/<id>/password</code> • retrievePasswordQualityRequirements (GET) from endpoint <code>/scim/v2/<id>/passwordQualityRequirements</code> • retrieveAccountState (GET) from endpoint <code>/scim/v2/<id>/account</code> • replaceAccountState (PUT) to endpoint <code>/scim/v2/<id>/account</code>
Account Operations	
	Only allowed if the SCIM Resource Type is User. Operations can include: <ul style="list-style-type: none"> • retrieve-consent (GET) from endpoint <code>/scim/v2/<id>/consents</code> or <code>/scim/v2/<id>/consents/<id></code> • revoke-consent (DELETE) from endpoint <code>/scim/v2/<id>/consents/<id></code> • retrieve-consent-history (GET) from endpoint <code>/scim/v2/<id>/consents</code> or <code>/scim/v2/<id>/consents/<id></code>
Consent Operations	

Resource Scope Properties	
Property	Description
	Only allowed if the SCIM Resource Type is <code>User</code> . Operations can include: <ul style="list-style-type: none"> <code>retrieve-external-identity</code> (GET) from endpoint <code>/scim/v2/<id>/externalIdentities</code> or <code>/scim/v2/<id>/externalIdentities/<id></code> This will expose access tokens from the identity provider. <code>unlink-external-identity</code> (DELETE) from endpoint <code>/scim/v2/<id>/externalIdentities/<id></code>
External Identity Operations	
	A list of one or more SCIM attributes of the authenticated identity for which this scope allows access. The type of access is determined by the operation properties <code>create</code> , <code>retrieve</code> , <code>replace</code> , and <code>modify</code> . A wildcard value of <code>*</code> can be used for all attributes. A schema-specific wildcard value of the form <code>urn:<schemaName>:*</code> can be used to represent all attributes of a single schema namespace. Access to attributes allowed per operation is the union of all <code>resourceAttributes</code> allowed in the scope.
Resource Attributes	

For granting access to Data Broker resources, the values of the `resourceAttributes` property are attribute notation strings as defined in the SCIM 2.0, with the addition of being able to specify wildcards for all attributes.

Note

The default OAuth2 Scope policy will deny requests for Resource scopes unless the [client credentials grant type](#) is used (or, if a different grant type is used and the end user has [the admin entitlement](#)).

Creating Scopes

An OAuth2 scope indicates which data are being requested with an OAuth2 authorization request. Typically, one or more scopes are submitted with each request. Scopes are created based on the access and authentication requirements of the data requested. A standard set of OpenID Connect scopes is installed with the Data Broker, and additional scopes can be created.

The following is a sample command for creating a scope:

```
$ bin/dsconfig create-oauth2-scope \
  --scope-name workPhone \
  --type authenticated-identity \
  --set "consent-prompt-text:Can I access your work phone number?" \
  --set consent-operation:retrieve-consent \
  --set external-identity-operation:link-external-identity \
  --set account-operation:retrieve-account-state \
  --set resource-attribute:work-phone \
  --set resource-operation:modify
```

Scopes can also be created in the Management Console through **Authorization and Policies -> OAuth2 Scopes**.

Creating an Authenticated Identity OAuth2 Scope

The following information is used to configure an Authenticated Identity scope. See [Authenticated Identity Scope](#) for details about the values allowed for consent, external identity provider, account, and resource operations.

- An OAuth2 access token name that is compliant with the OAuth 2.0 Specification (RFC 6749). The following characters are not permitted: space, "'", '\', '+', and ','.
- An optional description.
- Any optional tags associated with this scope. Tags are arbitrary additional properties that can be examined by XACML policies for authorization decisions, such as `HIPAA` or `billing`.
- The text displayed to a user when prompting for consent to access this scope.
- Specify the operations allowed by this scope on a [consent sub-resource](#).
- If performing authentication through an external identity provider, specify actions allowed by this scope on an external identity sub-resource.
- Specify the account management operations allowed by this scope. These actions rely on [configuration in the Data Store](#).
- Specify the resource attributes for which this scope allows access. The type of access is determined by the Resource Operation property. A value of "*" indicates that all attributes are accessible.
- Specify the operations allowed by this scope on the specified resource attributes.

Creating a Resource OAuth2 Scope

All of the Authenticated Identity values are available for the Resource scope, with the addition of the SCIM Resource Type that specifies the type of resource to which the scope provides access. See [Resource Scope](#) for details about the values allowed for consent, external identity provider, account, and resource operations.

Identity Provider Configuration

The Data Broker as an Identity Provider is responsible for providing token and authorization code identifiers for users trying to interact with the system. Authentication can be performed by the Data Broker, or the Data Broker can rely on a configured external identity provider. The following are configured for the Data Broker:

- [External Identity Provider](#) – Configure Facebook, Google Plus, or an OpenID Connect provider for authentication.

- [External Identity Provider Attribute Mapping](#) – Map External Identity Provider attributes to SCIM Resource Types, and define what should occur at login if there is a change to an attribute.
- [Identity Authenticator](#) – The default Username Password Identity Authenticator is configured, but can be changed, or another authenticator can be created. Identity Authenticators define how a user authenticates with the Data Broker.
- [Identity Provider Service](#) – Defines the properties that affect the Data Broker OAuth2 service including access token settings, password recovery options, account recovery options, and the use of reCAPTCHA.

Settings for the previous features are used to configure the following:

- [OAuth2 Client](#) – Define the OAuth2 clients that can request access to resources through the Data Broker.
- [OpenID Connect Claim](#) – OpenID Connect Claims define a claim that can be exposed through the UserInfo endpoint, and its mapping to attribute(s) of the identity resource.

Defining the Identity Provider Service

The Identity Provider Service defines the properties that affect the Data Broker OAuth2 functions. These settings are used to define the actions that the Data Broker can perform as an Identity Provider, including token rules and restrictions, log in and registration parameters, and username and password recovery restrictions.

The following is a sample command line configuration:

```
$ bin/dsconfig set-identity-provider-service-prop \  
--set "authorization-code-validity-duration:2 m" \  
--set "access-token-validity-duration:11 h" \  
--set "refresh-token-validity-duration:3 w 1 d" \  
--set register-enabled:true \  
--add register-resource-attribute:address \  
--set recover-password-enabled:true \  
--set recaptcha-key:<key> \  
--set recaptcha-secret:<secret>
```

The username and password recovery features are dependent on the configuration in the Data Store. See [Account Recovery Configuration in the Data Store](#) for details.

The Identity Provider Service can also be configured in the Management Console under **Identity Provider -> Identity Provider Service**. The following information is needed to configure the Identity Provider Service.

OAuth2 and OpenID Connect Settings

- Authorization code validity duration specifies the length of time an authorization code is valid. OAuth2 client configuration can specify a different validity duration that is specific to authorization codes generated for that client, which will override this property.

- Access token validity duration specifies the length of time an access token is valid. OAuth2 client configuration can specify a different validity duration that is specific to access tokens granted for that application, which will override this property.
- Refresh token validity duration. OAuth2 client configuration can specify a different validity duration that is specific to refresh tokens generated for that application, which will override this property. A value of 0 will disable the generation of refresh tokens.
- ID Token validity duration specifies the length of time an OpenID Connect token is valid. OAuth2 client configuration can specify a different validity duration that is specific to ID tokens granted for that client, which will override this property.
- ID Token issuer name specifies a unique identifier for the Issuer (`iss`) claim of an ID token. This value is inserted into a URL of the form `https://issuer_name` when returned as the unique issuer identifier in an OpenID Connect ID token. The default value for this property is the host name of the Data Broker installation.
- The signing algorithm to use when generating an OpenID Connect ID token. OAuth2 client configuration can specify a different signing algorithm that is specific to responses generated for that client, which will override this property.

General Settings

- SCIM Resource Type containing the attributes of identities that can be authenticated by the Data Broker, which must be configured with a primary LDAP store adapter connected to an UnboundID Data Store or an UnboundID Proxy Server. The Data Broker performs authentication against this SCIM Resource Type using the credentials provided through the login interfaces and REST APIs. Attributes of the authenticated identity can be retrieved and provided to clients through the SCIM `/Me` endpoint or OpenID Connect claims. Account management, password management, consent management, external identity provider login and linking, and self-registration are performed against identities in this SCIM Resource Type.
- The default login and logout URLs.
- The default [Identity Authenticator](#), which determines the authentication scheme for the Data Broker's Identity provider function.

Self-Service Account Flows

The following settings determine the features that are enabled for user registration, account recovery, and password reset. See [User Account Registration and Recovery](#) and [Account Recovery Configuration in the Data Store](#) for examples and additional requirements.

- **Register Enabled** – Specifies whether or not the register self-service account flow should be enabled. When disabled, the link will not be rendered on the login view and any attempts to access the register endpoint will result in a 403 Forbidden HTTP status code.

- **Register Resource Attribute** – Specifies the resource attribute paths that the registration account flow should allow the client to set. Registration will fail if a client submits a resource with attributes having paths that are not in this list.
- **Recover Username Enabled** – Specifies whether or not the username recover self-service account flow should be enabled. When disabled, the link will not be rendered on the login view and any attempts to access the username recovery endpoint will result in a 403 Forbidden HTTP status code. If enabled, the Data Store should be configured with an OTP (one time password) Delivery Mechanism and a `single-use-token` Extended Operation Handler.
- **Recover Username Search Filter** – Specifies the SCIM query used when the username recover self-service account flow searches for the account to recover. This SCIM filter expression should refer to attributes available in the SCIM Resource Type specified for Identity Provider Service. This filter should be constructed to not return more than a single result entry, and should not cause the primary store adapter to perform unindexed searches.
- **Recover Username Validity Duration** – Specifies the duration the username recover code is valid before expiring.
- **Recover Username Full Text** – Specifies the full text sent with the username recover code, if the one time password mechanism supports long text.
- **Recover Username Compact Text** – Specifies the compact text sent with the username recover code, if the one time password mechanism does not support long text.
- **Recover Username Subject** – Specifies the subject sent with the username recover code when the one time password mechanism supports subjects.
- **Recover Password Enabled** – Specifies whether or not the password recover self-service account flow should be enabled. When disabled, the link does not display on the login page and any attempts to access the password recovery endpoint will result in a 403 Forbidden HTTP status code. If enabled, the Data Store should be configured with an OTP Delivery Mechanism and a `deliver-password-reset-token` Extended Operation Handler.
- **Recover Password Search Filter** – Specifies the SCIM query used when the password recovery self-service account flow searches for the account to recover. This SCIM filter expression should refer to attributes available in the SCIM Resource Type specified for Identity Provider Service. This filter should be constructed to not return more than a single result entry, and should not cause the primary store adapter to perform unindexed searches.
- **Recover Password Full Text** – Specifies the full text sent with the password reset code, if the Data Store's one time password mechanism supports long text.

- **Recover Password Compact Text** – Specifies the compact text sent with the password reset code, if the Data Store's one time password mechanism does not support long text.
- **Recover Password Subject** – Specifies the subject sent with the password reset code when the one time password mechanism supports subjects.
- **Recaptcha Key** – Specifies the Google reCAPTCHA API key the register and recover self-service account flows should use. If a key is not specified, reCAPTCHA is not used.
- **Recaptcha Secret** – Specifies the Google reCAPTCHA API secret the register and recover self-service account flows should use. If a secret is not specified, reCAPTCHA will not be used by those flows.

Creating an Identity Authenticator

Identity Authenticators define how a user can authenticate with the Data Broker. A default Username Password Authenticator is enabled, and can be changed, or a new authenticator can be configured.

The following is a sample command:

```
$ bin/dsconfig create-identity-authenticator \
  --authenticator-name "New Authenticator" \
  --type username-password \
  --set enabled:true \
  --set 'match-filter:userName eq "$1"'
```

Configure the following for an Identity Authenticator:

- The a name and optional description for this authenticator.
- A match filter, which specifies the SCIM search filter that should be used when performing searches to map the provided username to a backend user resource. The filter pattern can include a string from a capturing group matched by the match pattern by using a dollar sign (\$) followed by an integer value that indicates which capturing group should be used. Capture group 0 refers to the entire username that matched. For example, the match-filter `userName eq $1` and `organization eq $2` with a match pattern of `^(.*)@(.*)$` will substitute \$1 and \$2 with the portions before and after the '@' symbol in the username.
- A match pattern, which specifies the regular expression pattern used to identify portions of a username. Any portion of the username that matches this pattern is replaced with the provided match-filter replace pattern. If multiple substrings within the given username match this pattern, all occurrences are replaced. If no part of the given username matches this pattern, the match-filter is not altered. It must be a valid regular expression as described in the API documentation for the `java.util.regex.Pattern` class, including support for capturing groups. For example, a match-pattern of `^(.*)@(.*)$` will match an e-mail address username. The match filter `userName eq $1` and

Chapter 4: Identity Provider Service and Scopes

`organization eq $2` can then be used to substitute \$1 and \$2 with the portions before and after the '@' symbol in the username.

Note Make sure that SCIM search functions are designed to return one, unique username. See Maintaining [Username Uniqueness](#) for details.

Chapter 5: User Authentication

The Data Broker supports the OpenID Connect Standard 1.0, which enables an OAuth2 client to use the Data Broker as its Identity Provider. OpenID Connect enables the client to offload its user authentication function to the Data Broker, which will prompt the end user for a login name and password and issue an ID Token that the client can use to validate the user's identity.

Obtaining an access tokens, refresh tokens, and token validation are fully documented in the OpenID Connect 1.0 specification.

Topics include:

[HTTP Authentication Schemes](#)

[OpenID Connect Request](#)

[OpenID Connect Response](#)

[Using the Data Broker as Relying Party](#)

[Creating an External Identity Provider](#)

HTTP Authentication Schemes

Three basic authentication schemes are provided for logging into or registering with the Data Broker. These schemes define the URLs that manage requests, produce failure pages, and determine where an end user is directed after successfully authenticating.

- Form Login HTTP Authentication Scheme – Used to provide a form login configuration for authentication with a username and password.
- External Identity Provider HTTP Authentication Scheme – Used to provide a login configuration for authentication with an external identity provider.
- Registration HTTP Authentication Scheme – Used to create a new user from attributes submitted using a form and then authenticate as that user.

OpenID Connect Request

To authenticate an end user, an OAuth2 client must have the following information from the Data Broker server administrator:

client identifier - An unique identifier issued to the client by the Data Broker server to identify itself.

client secret - A shared secret established between the Data Broker Server and the client application that is used for signing the ID token when it is returned to the client.

authorization, token, validate, endpoint URLs - The Data Broker's HTTP endpoint addresses for authenticating the end user, obtaining authorization, and issuing and validating access tokens. See [Data Broker Endpoints for OAuth2 Clients](#) for details.

userinfo endpoint - The address of the resource that, when presented with a token by the client, returns attributes about the end user.

The client uses this information to create an OAuth2 request to obtain an access token.

The following example request uses the implicit grant flow:

```
GET /authorize?response_type=token%20id_token&client_id6c7283d2-92d6-4767-9ceb-ada61e5e7e0d&state=4848573984983&scope=openid%20profile&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

An OAuth2 request becomes an OpenID Connect request with the inclusion of the `openid` scope. With the `openid` scope and the `response_type=id_token`, the client is requesting an identifier for the user as well as the ID token. The Data Broker [XACML policies](#) will determine the attributes that the clients can access within any scopes that are defined.

OpenID Connect Response

If the end user logged in properly and authorized the OAuth2 client request, the response from the Data Broker server includes an access token. If the request is an OpenID Connect request (contains the `openid` scope and `response_type=id_token`), the OAuth2 access token

response will include the `access_token` and `id_token` parameters. The following is encoded as a JSON Web token in the `id_token`:

aud (audience) – The client for which this token is intended.

exp (expiration) – The time after which this token is no longer valid.

iat (integer). The time at which the token was issued.

sub (subject) – A locally unique identifier for the end user. This value is never reassigned.

iss (issuer) – An HTTPS URI that is the fully qualified host name of the issuer, which is paired with the user identifier to create a globally unique identifier.

nonce – The `nonce` value sent in the request to ensure that the response is original and cannot be reused.

The `id_token` parameter ensures that the data received by the OAuth2 client has not been modified. The Data Broker can only issue assertions about registered clients and user identifiers within its domain. The token is validated by the Data Broker `/oauth/validate` endpoint. The client must do the following:

- Verify that the `aud` matches its client ID and `iss` matches the domain of the server that issued the client ID.
- Store the user identifier and `iss` together.

The following is an example of a base64url decoded ID token:

```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "auth_time": 1311280969,
}
```

The Data Broker as a Relying Party

The Data Broker, as relying party, acts as a client of an external identity provider service. Users can log into the Data Broker with external identity provider accounts. The Data Broker provides authentication claims, account linking, and profile retrieval services to the OAuth2 client. The Data Broker must be registered with the identity provider to enable this flow.

A social login link (and icon) can be configured and displayed on the Data Broker's default login page for clients configured to use an external identity provider. The login template reads this information through the `LoginPageContextProvider`. See [About Velocity Templates](#) for more information.

When an end user clicks an external identity provider link, a GET request is sent to the `/oauth/account/idpLogin` endpoint with the following two form parameters:

```
idp=<external identity provider name>
client_id=<requesting client id>
```

The `/oauth/account/idpLogin` endpoint redirects the browser to the external provider's authorization endpoint with an OpenID Connect code request:

```
response_type=code
client_id=<relying party client id>
redirect_uri=https://<host>:<port>/oauth/account/idpCallback/<externaIdpName>
state=<state value generated by the /oauth/account/idpLogin endpoint>
scope=<all scopes registered with the relying party client, including 'openid'>
```

The `<externaIdpName>` is the name given to this provider in the Data Broker. After the end user authenticates to the external identity provider and authorizes the OpenID Connect request, the external provider redirects the browser to the Data Broker's `/oauth/account/idpLogin` endpoint, as provided in the `redirect_uri` value. If a matching account is found at the Data Broker, the end user will need to log in to link the Data Broker account and the account at the external provider. Otherwise, a new Data Broker account can be created.

Note

The `redirect_uri` value used in this flow should be registered as a redirect URI with the application used by the Data Broker at the external identity provider. It should have the form `https://<DataBrokerServer>:<port>/oauth/account/idpCallback/<externaIdpName>`.

Creating an Account through Identity Provider Login

If an end user does not have a Data Broker account, one can be created by the Data Broker with the information obtained from the external identity provider.

The Data Broker applies the [SCIM Resource Type mappings](#) for the identity provider to the retrieved profile data. A 'Create and Link Account' form is displayed to the user. The user supplies the information, which is submitted to the SCIM `/registration.do` endpoint with the following parameters. If no additional information is needed, a new Data Broker account is created.

```
client_id=<requesting client id>
identityResource=<dynamically generated SCIM representation of the account to be created>
idpToken=<a token that contains state information about the authentication/registration request>
```

The user is redirected to the authorization URI specified by the requesting client, and the flow continues to the consent page for the scopes requested by the client. If the user consents, the client receives an access token issued by the Data Broker.

Creating an External Identity Provider

An External Identity Provider can be used to provide a social log in option to users, reset or retrieve account usernames and passwords, and link existing account data to data in a SCIM Resource Type. The options for these actions are defined in the [Identity Provider Service](#).

The Data Broker provides templates for creating Facebook and Google identity providers, or an OpenID Connect provider can be configured. All can be created through the Management Console or from the command line.

The following is a sample command for creating a Facebook identity provider:

```
$ bin/dsconfig create-external-identity-provider \
--provider-name "Facebook Provider" \
--type facebook \
--set "description:Facebook for Web App access" \
--set enabled:true \
--set app-id:847392057829512 \
--set "app-secret:AACbIFDY0ke7lyMjDhfBVsgYk+9BczWYM24=" \
--set permission:email
```

The following general information is needed to add any identity provider:

- A name and an optional description for this provider.
- The location URI for the icon that will represent the identity provider on the Data Broker login page.
- The hostname verification method for making sure the identity provider's hostname matches the name(s) stored inside the server's X.509 certificate. This is only needed if SSL is being used for connection security. Options are:
 - **allow-all** - Turns hostname verification off.
 - **strict** - Works like the Java Runtime Environment, and accepts wildcards. The hostname must match any of the Subject Alternative Names or the first CN. A wildcard can be present in the CN, and in any of the Subject Alternative Names. A wildcard such as *.example.com matches only subdomains in the same level, for example a.example.com. It does not match deeper nested subdomains.
- If using SSL, provide the location (DN) of the Key Manager and Trust Manager. If not provided, The Java Runtime Environment's default Key Manager and Trust Manager will be used.
- Attribute mapping for each provider defines how the value of a single SCIM Resource Type attribute is determined from an External Identity Provider attribute. The SCIM Resource Type is defined in the [Identity Provider Service](#).

For Facebook

- The App ID that was given to the Data Broker when it was registered with Facebook.
- The App Secret that was given to the Data Broker when it was registered with Facebook.
- Facebook permissions. These are the Facebook scopes that can be requested from a registered Data Broker OAuth2 client.

For Google

- The Client ID that was given to the Data Broker when it was registered with Google.
- The Client Secret that was given to the Data Broker when it was registered with Google.
- The Google scopes that can be requested from a registered Data Broker OAuth2 client.

For OpenID Connect

- The Client ID that was given to the Data Broker when it was registered with the identity provider.
- The Client Secret that was given to the Data Broker when it was registered with the identity provider.
- The OpenID Connect scopes that can be requested from a registered Data Broker OAuth2 client.
- Choose the authentication method to use when the OAuth2 client connects to the identity provider's token endpoint.
- The URL that the identity provider recognizes as its issuer identifier.
- The URL for the identity provider's OAuth2 authorization endpoint.
- The URL for the identity provider's OAuth2 token endpoint.
- The URL for the identity provider's OAuth2 UserInfo endpoint.

Chapter 6: OAuth2 Clients and Token Access

OAuth2 clients request access to scopes. Each request is processed by XACML policies, which determine whether the scope can be granted. Adding an OAuth2 client to the Data Broker defines the URL, the OAuth2 grant types, token requirements, and the scopes that the client can use. A client ID and client secret are defined and are needed by the OAuth2 client to interface with the `/oauth` endpoints. A redirect URL is needed during the registration process so that the Data Broker can redirect an end user back to the client when authorizing access to resources.

Topics include:

[OAuth2 Client Considerations](#)

[Using Applicable Scopes](#)

[Creating OAuth2 Clients](#)

[OAuth2 Authorization Grant Types](#)

[OAuth2 Authorization Response Types](#)

[Processing Access Tokens](#)

[The Data Broker Token Endpoint](#)

[Token Validation by the Data Broker](#)

[Token Revocation by the Data Broker](#)

[Obtaining a Refresh Token](#)

[Accepting External Access Tokens](#)

[The Data Broker Logout Endpoint](#)

OAuth2 Client Considerations

Consider the following when configuring an OAuth2 client to connect with the Data Broker:

- Assign only the [grant types](#) needed by the OAuth2 client. For example, it should be rare that a client needs to use both the code and the implicit grant types.
- The client should request only needed [scopes](#). Requesting only necessary information ensures that a user's privacy is respected and maintained.
- When a client receives an access token, it should not assume that all requested scopes were granted. The token response will often contain the list of granted scopes. In the case of the implicit grant type, the list of granted scopes will only be provided if they differ from the requested scopes. The validation endpoint can always be used to get the list of granted scopes.
- Access tokens granted using the implicit grant type should be configured to be short-lived.
- Access tokens should be validated to confirm that they are intended for the client.
- Any state information that must be preserved between requests should be stored using the `state` parameter. The `redirect_uri` value should not be used to store state.

OAuth2 Authorization Grant Types

The OAuth2 specification states that a client application must receive authorization from a resource owner through an access token, to retrieve the owner's protected resources. The Data Broker supports all OAuth2 authorization grant types:

- **Authorization Code Grant** – This is a server-side redirection-based flow. The OAuth2 client redirects the end user (user agent) to the authorization endpoint (Data Broker) to grant or deny access to a resource. If access is granted, the Data Broker returns a redirection URI with an authorization code and then redirects the end user back to the client. The client uses the authorization code to request an access token from the Data Broker server. The Data Broker validates the authorization code and returns an access and optionally a refresh token to the client. The client can now use the access token to request resources. The access token serves as both authentication of the client, and authorization to access the resources.
- **Implicit Code Grant** – This is another redirection-flow, designed for web clients, such as mobile applications or JavaScript applications running in browsers. The flow is similar to the authorization grant flow, except that the Data Broker redirects the client with an embedded access token in the URI, rather than an authorization code requiring a separate token request. The client secret is not used because it would be stored (and be

vulnerable) in the client. No refresh token is sent as this grant type is designed for short-lived access to a resource.

- **Resource Owner Password Credentials Grant** – This flow enables a user to log in with a username and password to receive an access token. The OAuth2 client can then keep the access token for access to resources. The client is expected to discard the username and password and keep the access token. This flow should only be used for clients that are trusted to handle the user password in the clear, as well as detailed account and credential validation errors.
- **Client Credentials Grant** – This flow enables a client's application server to exchange the client ID and the client secret for an access token. This enables clients to directly access resources that are specific to the client and are not tied to an identity.
- **ID Token Grant** – This enables a set of trusted clients to allow one application to use an OpenID Connect ID token, obtained by another application, as a credential for obtaining an access token on behalf of an end user.

OAuth2 Authorization Response Types

The Data Broker supports the following OAuth2 and Open ID Connect response types:

- `code` – to request an authorization code.
- `token` – to request an access token.
- `token id_token` – to request both an access token and an ID token.

Issuing Authorization Code Grant Requests

The Authorization Code Grant flow follows these basic steps:

1. Redirect the user agent (end user) to the Data Broker's authorization endpoint.
2. Resource owner authenticates and grants authorization.
3. Data Broker redirects the user to a web application with an authorization code.
4. The authorization code is exchanged for an access token.
5. A request to access resources is sent to the Data Broker using the access token.

Step 1. Redirect the User Agent to the Data Broker's Authorization Endpoint

The OAuth2 client requires access to a protected resource and needs an access token that represents the required permissions. The client redirects the end user to the Data Broker's authorization endpoint (`/oauth/authorize`). The HTTP request URL includes the `response_type=code`, the `client_id`, and optional values for the `redirect_uri` specifying the redirect URL to redirect.

Example Redirection

```
GET /oauth/authorize?response_type=code&client_id=0d5e5af7-420c-4241-8cff-0cfd9d806e59&scope=profile%20email&state=48389488&redirect_uri=https%3A%2F%2Fwww.example.com%3A8443%2Fredirect&prompt=login HTTP/1.1
Host: <server.example.com>
```

Step 2. Resource Owner Authenticates and Grants Authorization

The authorization request is run through XACML policies. If a policy rule results in a denial, an error is generated. If the authorization request passes the policy rules, the resource owner is sent a Data Broker web page to provide credentials and consent, if not previously provided.

Step 3. Data Broker Redirects User Agent to Web Application with Authorization Code

If the resource owner has granted access to the OAuth2 client, the Data Broker redirects the user back to the client web application and includes an authorization code that can be exchanged for an access token.

Example Response

```
HTTP/1.1 302 Found
Location: https://<server2.example.com>?code=MF2AAQGBB1pxSGUtUYJQo2oB1p1kw3CNCm5QRmok-vzKYVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKm4QVx6mCTmT9gztIn45K9KKJ22p8IiJHiLXGEg2oUV&state=48389488
```

Step 4. Exchange Authorization Code for an Access Token

The OAuth2 client posts a request to the token endpoint (Data Broker) to acquire an access token. This step is not performed by the browser. The client request must supply the `client_ID` and `client_secret` using HTTP Basic authentication.

Example Request

```
POST /oauth/token HTTP/1.1
Host: <server.example.com>
Authorization: Basic czQER9k3dD94aIdplr957Udk8
Content-Type: application/w-www-form-urlencoded

grant_type=authorization_code&code=MF2AAQGBB1pxSGUtUYJQo2oB1p1kw3CNCm5QRmok-vzKYVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKnav_aXvvyuxT3ogtZT-dgNZEnk6X0XaoPf6BVlVRibA&redirect_uri=https%3A%2F%2Fserver2%2Eexample%2Ecom
```

The Data Broker validates the authorization code and verifies that the `redirect_uri` is the same as in Step 1. The response may include a refresh token and/or an ID token, depending on the request. If successful, the server issues the following response:

Example Response

```
HTTP/1.1 200 OK
Cache-Control: no-store
Pragma: no-cache
```

```
Content-Type: applicaton/json;charset=UTF-8
Transfer-Encoding: chunked
Server: Jetty(8.1.12.v20130726)

{
  "access_token":"MF2AAQGGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-
vzKYVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKMYeiJy-24paR9YLEZpKDc-mw1E4ML8LRqAyhPMtAoBA",
  "token_type":"bearer",
  "expires_in":41558,
  "scope":"email profile"
}
```

Step 5: Request Access to the Resources Using the Access Token

The client can now query the Data Broker (acting as the Resource server) for a restricted resource by passing along the access token in the authorization header of the request.

Example Request

```
GET /scim/resource HTTP/1.1
Host: server.example.com
Authorization: Bearer MF2AAQGGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-
vzKYVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKMYeiJy-24paR9YLEZpKDc-mw1E4ML8LRqAyhPMtAoBA
```

The Data Broker returns the requested information.

Issuing Implicit Code Grant Requests

The Implicit Code Grant flow follows these basic steps:

1. Redirect the user agent (end user) to the Data Broker's authorization endpoint.
2. Resource owner (end user) authenticates and grants authorization.
3. Redirect user agent to a web application with a URI fragment containing the access token.
4. Client-side web application responds with an HTML page with a script that retrieves the access token from the URI fragment.
5. Request access to resources using access token.

Step 1. Redirect the User Agent to the Data Broker's Authorization endpoint

The client redirects the end user to the Data Broker's authorization endpoint. The HTTP request URL includes the `response_type=token`, the `client_id`, which was determined when the client was added to the Data Broker, the `redirect_uri`, and `scope`.

Example Redirection

```
GET /oauth/authorize?response_type=token&client_id=6c7283d2-92d6-4767-9ceb-
ada61e5e7e0d&state=4848573984983&redirect_
uri=https%3A%2F%2Fserver2%2Eexample%2Ecom&scope=profile%20email HTTP/1.1
Host: <server2.example.com>
```

Step 2. Resource Owner Authenticates and Grants Authorization

The authorization request is run through XACML policies. If a policy rule results in a denial, an error is generated. If the authorization request passes the policy rules, the resource owner is sent a Data Broker web page to provide credentials and consent if not previously provided.

Step 3. Redirect User Agent to Web Application with Access Token URI Fragment

Once the resource owner has granted access rights to the OAuth2 client, the Data Broker sends a redirect response, sending the user back to the client (web application). The redirect URI includes an access code in the `#hash` fragment of the URI.

Example Redirect Response

```
HTTP/1.1 302 Found
Location: https://<server2.example.com>/callback#access_
token=1MF2AAQGGB1pxSGUtUYJQo2oB1pkw3CNcM5QRmok-
vzKYVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKMYeiJy-24paR9YLEZpKDC-
mw1E4ML8LRqAyhPMtAoBA&token_type=bearer&state=4848573984983&expires_in=43062
```

Step 4. Client-Side Web Application Responds with an HTML Page

The user agent (browser) is redirected to the URL and the client responds by serving an HTML page containing scripts to parse the access token from the URI. If a state value is present, the script should evaluate the parameter.

Step 5: Request Access to the Resources Using the Access Token

The OAuth2 client can now query the Data Broker (as Resource server) for resources by passing along the access token in the authorization header of the request.

Example Request

```
GET /scim/resource HTTP/1.1
Host: <server.example.com>
Authorization: Bearer MF2AAQGGB1pxSGUtUYJQo2oB1pkw3CNcM5QRmok-
vzKYVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKMYeiJy-24paR9YLEZpKDC-mw1E4ML8LRqAyhPMtAoBA
```

The Data Broker returns the requested information.

Issuing Resource Owner Password Credentials Requests

The Resource Owner Password Credentials Grant flow follows these basic steps:

1. Client asks for the resource owner's (end user's) credentials.
2. Client makes an authorization request to the Data Broker's token endpoint (`/oauth/token`).
3. Client receives the access token.
4. Request access to resources using the access token.

Step 1. Client Asks for Resource Owner's Credentials

The OAuth2 client prompts for the resource owner's username and password when the application requires access to resources that are protected by the Data Broker, but has not yet acquired an access token. This flow should only be used for trusted clients.

Note

If requested, account usability notices, warnings, and errors for a user can be returned in the token endpoint responses when using the resource owner password credentials grant type. This may be helpful for a trusted OAuth2 client to validate elements of an authentication flow. For example, the client can check for account usability errors in a successful token response to see if it needs to prompt a user for password changes.

Step 2. Client Makes an Authorization Request at Token Endpoint

The OAuth2 client makes an authorization request to the Data Broker's token endpoint by passing in the `client_id` and `client_secret` and the resource owner's username and password. The client credentials must be passed through a basic authentication request header.

Example Request

The following HTTP request uses basic authentication with the `client_id` and `client_secret`, concatenated, encoded, and separated by a colon. The format is:

Authorization: Basic <Base64-encoded client_id:client_secret>

```
POST /oauth/token
Host: <server.example.com>
Authorization: Basic czQER9k3dD94aIdplr957Udk8
Content-Type: application/w-www-form-urlencoded

grant_type=password&username=johndoe&password=A3ddj3w
```

If the request is valid, the Data Broker returns an access token (and possibly a refresh and/or ID token) to the client. Once the client receives the response, it should discard the resource owner's username and password.

Example Response

```
HTTP/1.1 200 OK
Cache-Control: no-store
Pragma: no-cache
Content-Type: applicaton/json;charset=UTF-8
Transfer-Encoding: chunked
Server: Jetty(8.1.12v20130726)

{
  "access_token": "MF2AAQGBB1pxSGUtUYJQo2oB1p1kw3CNcM5QRmok-
vzKYV1tlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKFEDrIpaEn5N9MfAm1BjZ5OYLHu0L823L2JsMn7i2wug",
  "token_type": "bearer",
  "expires_in": 42203,
  "scope": "profile",
}
```

Issuing Client Credentials Requests

The client credentials grant flow follows these basic steps:

1. Client makes an authorization request to the Data Broker's token endpoint.
2. Client receives the access token.

Step 1. Client Makes an Authorization Request at Token Endpoint

The OAuth2 client makes an authorization request to the Data Broker's token endpoint by passing the `client_id` and `client_secret`. The client credentials must be passed through a basic authentication request header.

The following HTTP request uses basic authentication with the `client_id` and `client_secret`, concatenated, and encoded.

Example Request

```
POST /oauth/token HTTP/1.1
Authorization: Basic amFiYmVyd29ja3k=
Content-Length: 41
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: server.example.com

grant_type=client_credentials&scope=email
```

Step 2. Client Receives the Access Token

If the request is valid, the Data Broker returns an access token. If the access token expires, the client credentials grant can be rerun to obtain a new access token.

Example Response

```
HTTP/1.1 200 OK
Cache-Control: no-store
Pragma: no-cache
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Server: Jetty(8.1.12v20130726)

{
  "access_token": "MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-
vzKYVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKFEDrIpaEn5N9MfAm1BjZ5OYLHu0L823L2JsMn7i2wug",
  "token_type": "bearer",
  "expires_in": 42203,
  "scope": "profile",
}
```

Considerations for using this Grant Type

The default OAuth2 Consent policy will deny requests for generic or [Authenticated Identity Scopes](#) if the client credentials grant type is used. Those scope types are expected to require

user consent, and the client credentials grant type does not involve an interaction with an end user.

The default OAuth2 Scope policy will deny requests for [Resource Scopes](#) unless the client credentials grant type is used (or, if a different grant type is used and the end user has [the admin entitlement](#)).

Access tokens obtained using the client credentials grant type are called 'application tokens,' since they are granted on behalf of an OAuth2 client instead of an end user. Application tokens are not persisted to the Data Broker configuration or the backend User Store. They are entirely self-contained, and the Broker can interpret them without having to retrieve information from a backend User Store.

An individual application token cannot be revoked using the Data Broker's token revocation endpoint, and should be used with care. The client credentials grant type should only be used by trusted applications, and the access token validity duration should be short. See [Defining the Identity Provider Service](#) for token configuration.

Issuing ID Token Grant Requests

A set of cooperating, trusted clients can use the ID Token Grant type to allow one client to use an OpenID Connect ID token, obtained by another client, as a credential for obtaining an access token on behalf of an end user. This enables a set of non-web-based applications to obtain access tokens for a particular end user, without the need for repeated prompts the end user for credentials. This grant type is based on the *JSON Web Token (JWT) Profile for OAuth2 Client Authentication and Authorization Grants (draft-ietf-oauth-jwt-bearer-12)* specification, but is not intended to conform strictly to that spec.

A request made using the ID Token Grant type is similar to a request made using the [Resource Owner Password Credentials Grant](#) type. However, the `username` and `password` parameters are replaced by an `assertion` parameter, with a valid JWT ID token as its value.

To use this grant type:

- All OAuth2 clients involved must be registered with the Data Broker to use the ID Token grant type.
- A shared ID token can be obtained using any standard OpenID Connect grant type. In addition, the Data Broker allows a client using the Resource Owner Password Credentials Grant type to obtain an ID token by using a `response_type` parameter with a value of `id_token`.
- An ID token obtained by a client that is not registered to use the ID Token Grant type cannot be used to make an ID Token Grant type request. Conversely, any OAuth2 client registered to use the ID Token Grant type may use an ID token obtained by any other client registered to use the ID Token Grant type.
- All clients should be secure and highly trusted. It is the responsibility of the clients to make sure that ID tokens are stored and shared in a secure manner.

The following example shows an OAuth2 client using the Resource Owner Password Credentials Grant type to obtain an access token and an ID token. The client provides a `response_type` parameter with a value of `id_token`, which instructs the server to return an ID token. (The response has been shortened in the example.) Once the token is received, the client must be able to securely store it and share it with other trusted clients.

The following example shows how a second, trusted client might then use the ID token grant type to obtain its own access token. A `grant_type` value of `unboundid_id_token` is used, and the ID token is provided as the value of the `assertion` parameter.

- 77 -

```

Pragma: no-cache
Server: Jetty(8.1.16.v20140903)
Transfer-Encoding: chunked

{
  "access_token":
"AVCGkIwEoDOKeQotjBQ8gVZvNq4HAAAAAAAAAAdr3dRrArNA53ANjXsFfu686hNNQ8ZN2iOtky2tQun0g7Z1VgRrK
AcfjQ62caZYbyzt9pKrLcCljtJwhzybz6KjKLd8Ma85gywk36Z4jTEMhjg",
  "expires_in": 43199,
  "scope": "profile",
  "token_type": "bearer",
}

```

Adding an OAuth2 Client

Create and maintain OAuth2 clients that can request access to resources based on XACML policy, or any other privacy restrictions. The information used to register the client with the Data Broker will be needed by the OAuth2 client to request resources. Clients can be added through the Management Console or from the command-line, such as:

```

$ bin/dsconfig create-oauth2-client \
  --client-name "Web App Client"
  --set scope:email

```

The information used to configure an OAuth2 client includes:

- The name of the client.
- Optional description, and contact email address for this OAuth2 client.
- The client ID and client secret can be generated by the Data Broker when the OAuth2 client is created, or they can be entered manually.
- The OAuth2 access grant types, which include:
 - **authorization-code** - The authorization code grant, which is used to request an access token from an authorization code.
 - **client-credentials** - The client credentials grant, which can be used by a client to request an access token using only its client credentials.
 - **implicit** - The implicit grant, where an access token can be requested without obtaining intermediate credentials (such as an authorization code).
 - **password** - The password grant, where an access token can be requested directly from the resource owner credentials.
 - **refresh-token** - The refresh token grant, where a new access token can be requested from a refresh token.
 - **unboundid-id-token** - ID token grant, where an access token can be requested using an ID Token assertion as authentication.
- Access token duration, and consent requirements.

- The client URL and any redirect URIs, which must also be registered with the Data Broker.
- The [scopes](#) that can be requested by an OAuth2 client.
- (missing or bad snippet)
- The [trusted origin\(s\)](#) of the client if making JavaScript requests.
- Any [external identity providers](#) that can be used to authenticate an end user account (Advanced setting).
- If necessary, the access and authentication token settings can be specified per client. If not specified when creating the OAuth2 client, the Identity Provider Service settings are used.

The Data Broker Token Endpoint

An OAuth2 client uses the token endpoint (`/oauth/token`) to obtain an access token by presenting its authorization grant. The endpoint can also issue a refresh token if the original access token has become invalid or expires. The authorization header of the client request will contain the Base64 encoded `client_ID` and `client_secret` credentials.

Note

The token endpoint can return errors, warnings, and notices related to the login identity's password and account state when using the [Resource Owner Password Credentials Grant type](#).

Request

The following example makes a token request to the endpoint:

```
POST /oauth/token HTTP/1.1
Host: <example.com>
Authorization: Basic aXQncyBkYW5nZXJvdXMgdG8gZ28gYWxvbmU6dGFrZSB0aGlz
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=Sp1xl0BeZQQYbYS6WxSbIA&redirect_
uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

Response

If the token request is authorized, the Data Broker server returns:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token": "2YotnFZFEjrlzCsicMWpAA",
  "token_type": "bearer",
  "expires_in": 3600,
```

```

"scope": "openid email profile",
"id_token": "eyJhbGciOiJIUzI1NiJ9.eyJhdXRoX3RpbWUiOjE0MjE4ODExMDMsImV4
cCI6MTQyMTg4MjAwOSwic3ViIjoioWY4YTlzlWNjY2M3NmVlLWQwN2ItM2I
4Yy05MjJjLWRkZDgwOWM0YzE3MyIsImF1ZCI6WyJhY211Ii0sImIzcyI6Im
h0dHBzOlwvXC94MjI1MC0wMMS5leGFtcGxlLmNvbSIsImIhdCI6MTQyMTg4M
TEwOX0.CZYpxocXZ-_DEPtHqSiQ1FU8Pplb8I-7oK3PMp4-Y"
}

```

Token Validation by the Data Broker

The Data Broker token validation endpoint (`/oauth/validate`) enables OAuth2 clients and external resource servers to determine the state of an access token, as well as additional metadata about the token. To validate an access token, a POST is sent to the Data Broker's `/oauth/validate` endpoint, which returns a response with information about the token's validity and scope. The validation endpoint is based on the OAuth 2.0 Token Introspection standard, RFC 7662.

Parameters are provided either as form parameters or as query parameters appended to the token validation endpoint URL. Though, using query parameters is discouraged because it will cause the access token to be logged.

The `token` parameter is required. A `client_id` parameter is optional. If both are provided, the validation endpoint verifies that the access token was issued to the provided client.

The token response includes a `jti` claim (JWT ID), which provides a unique identifier for the access token. The `jti` value also appears in the Data Broker's trace log output, and can be used to find requests using this access token.

Note

OAuth2 clients using OpenID Connect are responsible for validating ID tokens received from the Data Broker. Refer to the OpenID Connect Core 1.0 specification for information.

Request

The following is a request to validate a token:

```

POST /oauth/validate HTTP/1.1
Accept: application/json
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: example.com:443

token=<access token>&client_id=<client ID>

```

Response

If the operation is successful, the Data Broker responds with a JSON object with the following parameters:

```

HTTP/1.1 200 OK
Cache-Control: no-store
Pragma: no-cache
Content-Type: application/json; charset=UTF-8

{

```

```
{
  "active": true,
  "sub": "Users/1d998887-87bc-4271-aa0c-27652bf02d6c",
  "client_id": "<client ID>",
  "exp": 1448008233,
  "iat": 1447971141,
  "scope": "openid profile email"
  "jti": "IPaSog"
}
```

Token validation failures occur if the token is malformed, expired, or revoked. Failures will also occur if the provided `client_id` does not match the application for which the access token was issued. If validation fails, the response will indicate that the token is inactive:

```
HTTP/1.1 200 OK
Content-Length: 16
Content-Type: application/json;charset=UTF-8

{
  "active": false
}
```

Token Revocation by the Data Broker

The token revocation endpoint (`/oauth/revoke`) enables OAuth2 clients to send a POST request to the Data Broker to revoke access or refresh tokens. Revoking a token does not remove any associated consents.

During the revocation process, the Data Broker validates the client credentials, and verifies that the client making the request originally issued the token. If the validation fails, the request is refused and an error response is sent. If validation is successful, the Data Broker revokes or invalidates the token.

For example, the following revokes a token:

```
Authorization: Basic MC2AAQGBBlpxSGUtUYIgQI8F1rTZdspnJxDamsIKKxei8Wdj_E3DUXscVpiw6u8
POST /oauth/revoke HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Token=MC2AAQGBBlpxSGUtUYIgQI8F1rTZdspnJxDamsIKKxei8Wdj_E3DUXscVpiw6u8
```

If the operation is successful, the Data Broker responds with the HTTP status code 200.

The revocation endpoint requires HTTP Basic authentication using the `client_id` and `client_secret`, just like the `/oauth/token` endpoint.

Obtaining a Refresh Token

To request an OAuth2 refresh token, the `offline_access` scope should be requested in the client's authorization request. The client application's use and consent requirements will dictate the choice of scope:

The `offline_access` scope is provided for compliance with the OpenID Connect specification. To successfully obtain a refresh token, a client using this scope must also specify the prompt

authorization request parameter with a value of `consent`. End users must provide explicit consent to grant a refresh token every time one is requested.

Refresh tokens can only be requested with an authorization code grant request or a resource owner password credentials grant request. For example:

```
GET /oauth/authorize?
response_type=code& client_id=<0d5e5af7-420c-4241-8cff-0cfd9d806e59&
scope=profile%20email%20offline_access&
prompt=consent&
state=48389488& redirect_uri=https%3A%2F%2Fwww.example.com%3A8443%2Fredirect
```

The refresh token will be provided in the `refresh_token` field of the token response. The client may use a refresh token to extend the duration of an authorization without end user interaction by making a refresh request to the token endpoint to obtain a new access token. The following POST parameters are used:

- `grant_type` – Required. Value must be set to `refresh_token`.
- `refresh_token` – Required. The refresh token issued to the client.
- `scope` – Optional. The scope of the access request. The requested scope cannot include any scope not originally granted by the resource owner.

The response will look like the following:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token": "VGhlIGFwcGFyaXRpb24gb2YgdGhlc2UgZmFjZXMgaW4gdGhlIGNyY3dkOw==",
  "refresh_token": "UGV0YWxzIG9uIGEd2V0LCBibGFjayBib3VnaC4=",
  "token_type": "bearer",
  "expires_in": 3600,
  "scope": "profile email"
}
```

Accepting External Access Tokens

As a Resource server, the Data Broker supports receipt of access tokens from a third-party OAuth2 provider. Authorization of requests that use externally-defined access tokens require a custom [Policy Information Provider \(PIP\)](#) to interpret the capabilities granted to the token, which can be created with the [UnboundID Server SDK](#).

Access tokens must be validated by XACML policy. The Token Policy Information Provider is responsible for determining the capabilities of internally-issued access tokens and making them available to the policy engine.

If an externally-issued access token is presented with a request, the Token Policy Information Provider will be unable to interpret it. The policy engine in that case will continue through its list of configured PIPs until it finds one that can decode the token.

Though not required, a third-party PIP can offer the same JSON interface as the Data Broker's PIP, therefore making the type of token transparent to policy. This type of PIP must at least return a JSON object populated with the `active`, `sub`, and `scope` properties. A third-party PIP that provides a different interface would require policies to be written with specific knowledge of different token types.

The Data Broker Logout Endpoint

A POST to the `logout.do` endpoint will invalidate a user's session with the Data Broker and revoke the user's access tokens with either a single client or all clients registered with the Data Broker. The `client_id` and `post_logout_redirect_uri` query parameters are both optional.

If a `client_id` is not provided, all of the user's access tokens will be revoked. If a `client_id` is provided, then only the access tokens for that client are revoked.

If a `post_logout_redirect_uri` is not provided, the browser will be redirected to the configured `default-logout-success-url` for the [Identity Provider Service](#) (which defaults to `/view/login`). If a `post_logout_redirect_uri` is provided, then `client_id` must also be provided. The `post_logout_redirect_uri` value must match one of the redirect URIs configured for the client (which is retrieved by the `client_id`). The browser will be redirected to the provided `post_logout_redirect_uri` after logout.

Request

The following is an example POST to the Data Broker `logout.do` endpoint:

```
POST /logout.do?client_id=385b45d0-88bd-4973-a9bc-06484ad27e42&redirect_
uri=https://example-app.com/
Host: example.com
Content-Length: 0
Cookie: JSESSIONID=xpdpr7z6fxh31rjdpvgcmce0c
```

Response

The following is an example response:

```
HTTP/1.1 302 Found
Location: https://example-app.com/
Content-Length: 0
```

Chapter 7: Accessing Data

The Data Broker server supports two user profile endpoints:

- The SCIM endpoint provides full operations on user profile data through the SCIM protocol. The endpoint's URL context path is `/scim/v2/{name}`. Each SCIM resource, specified in the SCIM Schema, is exposed as an endpoint. For example, the URL path `/scim/v2/Users` would be used to access the `Users` SCIM resource.
- The OpenID Connect UserInfo endpoint enables the Data Broker to function as a Resource server. The endpoint's URL context path is `/userinfo`. The UserInfo endpoint is read-only and uses GET actions to retrieve user profile data.

Access to resources is determined by the XACML policies that are configured for the Data Broker. If a request to the Data Broker is delivering partial results, it may be due to policy settings. See [How Policy Affects Access to Scopes](#).

Topics include:

[Data Broker Endpoints for OAuth2 Clients](#)

[The SCIM Endpoint](#)

[SCIM Examples](#)

[UserInfo Access Example](#)

Data Broker Endpoints for OAuth2 Clients

The Data Broker provides multiple REST endpoints for client access. The following list presents a summary of the endpoints that may be called by a client application requesting user profile data. All Data Broker endpoints are available at `<server-root>/docs/restapi/index.html`.

A request to each endpoint should have a scope with the desired actions included. Review the properties and values available for [Authenticated Identity Scopes](#) and [Resource Scopes](#).

Data Broker Endpoints for Clients	
Endpoint	Description
/scim	
	This is the SCIM 2.0 protocol endpoint used to retrieve a specified SCIM Resource Type, where <code><name></code> is the SCIM Resource Type being accessed. This endpoint supports all SCIM operations and implements its access control through the XACML policies. A request to this endpoint requires a scope that includes a <code>resourceOperations</code> value that represents the desired action.
<code>/scim/v2/<name></code>	
/oauth	
<code>/oauth/authorize</code>	The OAuth2 standard authorization endpoint. This is the endpoint that an application will use to get an authorization grant from the user.
<code>/oauth/token</code>	The OAuth2 token endpoint. This is the endpoint that an application will use to request an access token from the Data Broker Server to access identity information.
<code>/oauth/revoke</code>	The Data Broker endpoint used to revoke a token.
<code>/oauth/validate</code>	The Data Broker endpoint used to validate a token.
/userinfo	
<code>/userinfo</code>	The OpenID Connect endpoint. Use this endpoint for applications that require read-only access to user profile data. Access to this endpoint requires an OAuth2 access token with the <code>openid</code> scope. The client application will receive the attributes granted by the scopes in the access token. Either GET or POST actions can be used.
/pdp/v1/authorization	
<code>/pdp/v1/authorization</code>	The Data Broker Policy Decision Point endpoint used by an external Policy Enforcement Point (PEP) to generate XACML requests and send them directly to the Data Broker for evaluation. The request is passed directly to the policy engine. This method supports POST only. The body of the POST should contain the XACML request as an XML string.

The SCIM Endpoint

The Data Broker SCIM endpoint enables applications to perform actions on an end user's resources, if XACML policies permit. The following are important to consider when using the SCIM endpoint:

/Me. SCIM supports a special endpoint to retrieve attributes of the currently authenticated user without knowing the SCIM ID. Retrieve attributes of the currently authenticated user with the following:

```
/scim/v2/Me
```

Authentication. The SCIM endpoints are protected by bearer token authentication, obtained from the Data Broker. See [Authentication](#) for details.

Note

/Bulk and /Groups are not supported.

SCIM Examples

A client application accesses the `/scim/v2/{name}` endpoint by passing an HTTP GET, POST, PATCH, or DELETE request with an access token parameter to the Data Broker Server. The response is a JSON object.

GET

The following is an example call to the Data Broker `/scim/v2/{name}` endpoint to get entries with the filter of user name starting with `sam`.

Note

A GET operation should not be performed with 'sensitive' attributes.

Request

```
GET /scim/v2/Users?startIndex=1&count=10&filter=userName+sw+%22sam%22
Host: example.com
Accept: application/json
Authorization: Bearer MF2AAQGGB1Y1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-LHJGskZwAd3dYWPVERzIAy-VczegSxSR2c5luoiFgSyQFfC_y0kLy15L4iTI
```

Response

The data returned is dependent on the Data Broker configuration and the XACML policies in place.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...
{
  "totalResults":1,
```

Chapter 7: Accessing Data

```
"schemas": [
  "urn:ietf:params:scim:api:messages:2.0:ListResponse:schema",
],
"Resources": [
  {
    "name": {
      "givenName": "Sample",
      "familyName": "User1",
      "formatted": "Sample User1"
    },
    ...// other user properties
  },
  ...// other users
]
```

jQuery Example

```
$.ajax({
  type: "GET",
  url: "https://example.com/scim/v2/Users",
  headers: { "Authorization": "Bearer " + accessToken },
  dataType: "json",
  success: function(usersPage) {
    // application can do something with returned data...
  }
});
```

GET (by User ID)

The following is an example call to the Data Broker `/scim/v2/{name}` endpoint to get a single user entry with the ID of `9f8a23-47c7be45-0ce5-3105-8ea8-fc3c39c47f91`.

Request

```
GET /scim/v2/Users/9f8a23-47c7be45-0ce5-3105-8ea8-fc3c39c47f91
Host: example.com
Accept: application/json
Authorization: Bearer MF2AAQGGB1Y1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-LHJGskZwAd3dYWPVERzIAy-VczegSxSR2c5luoiFgSyQFFC_y0kLy15L4iTI
```

Response

The data returned is dependent on the Data Broker configuration and the XACML policies in place.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:ListResponse:schema",
  ],
  "name": {
```

```

    "givenName": "Sample",
    "familyName": "User1",
    "formatted": "Sample User1"
  },
  ... // other user properties
}

```

jQuery

```

$.ajax({
  type: "GET",
  url: "https://example.com/scim/v2/Users/"+userId,
  headers: { "Authorization": "Bearer " + accessToken },
  dataType: "json",
  success: function(user) {
    // application can do something with returned data...
  }
});

```

POST

The following is an example call to the Data Broker `/scim/v2/{name}` endpoint that creates a user entry for Another Sample User III.

Request

```

POST /scim/v2/Users
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer MF2AAQGBBlY1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-LHJGskZwAd3dYWPVERzIAy-VczegSxSR2c5luoiFgSyQFfc_y0kLy15L4iTI
Content-Length: ...
{
  "schemas": [ "urn:ietf:params:scim:api:messages:2.0:ListResponse:schema" ],
  "name": {
    "formatted": "Another Sample User III",
    "familyName": "User",
    "givenName": "Another",
    "middleName": "Sample"
  },
  "userName": "sampleuser3"
}

```

Response

The data returned is dependent on the Data Broker configuration and the XACML policies in place.

```

HTTP/1.1 201
Created Content-Type: application/json
Content-Length: ...
{
  "schemas": [

```

Chapter 7: Accessing Data

```
    "urn:ietf:params:scim:api:messages:2.0:ListResponse:schema",
  ],
  "name": {
    "givenName": "Another",
    "familyName": "User",
    "formatted": "Another Sample User III"
  },
  "id": "9f8a23-3562ddf5-50d0-4aac-a761-7ecb9bcb7633",
  "userName": "sampleuser3",
  "meta": {
    "created": "2014-09-04T19:06:22.547Z",
    "lastModified": "2014-09-04T19:06:22.547Z",
    "location": "https://example.com/scim/v2/Users/9f8a23-3562ddf5-50d0-4aac-a761-7ecb9bcb7633"
  }
}
```

jQuery Example

```
$.ajax({
  type: "POST",
  url: "https://example.com/scim/v2/Users",
  data: JSON.stringify({
    "schemas": [ "urn:ietf:params:scim:api:messages:2.0:ListResponse:schema" ],
    "name": {
      "formatted": "Another Sample User III",
      "familyName": "User",
      "givenName": "Another",
      "middleName": "Sample"
    },
    "userName": "sampleuser3"
  }),
  headers: { "Authorization": "Bearer " + accessToken },
  contentType: "application/json",
  dataType: "json",
  success: function(user) {
    // returned data sample...
  }
});
```

UPDATE

The following is an example call to the Data Broker `/scim/v2/{name}` endpoint that updates a user entry for ID `9f8a23-31c5b68d-2c8d-4dd2-987b-09627cb1ff2d`.

Request

```
PATCH /scim/v2/Users/9f8a23-31c5b68d-2c8d-4dd2-987b-09627cb1ff2d
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer MF2AAQGGB1Y1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-LHJGSkZwAd3dYWPVERzIAy-VczegSxSR2c5luoiFgSyQFfC_y0kLy15L4iTI
Content-Length: ...
```

```
{
  "schemas": [ "urn:ietf:params:scim:api:messages:2.0:ListResponse:schema" ],
  "name": {
    "formatted": "My Sample Tester III",
    "familyName": "Tester",
    "givenName": "My",
    "middleName": "Sample"
  }
}
```

Response

```
HTTP/1.1 204 No Content
```

jQuery Example

```
$.ajax({
  type: "PATCH",
  url: "https://example.com/scim/v2/Users/"+userId,
  data: JSON.stringify({
    "schemas": [ "urn:ietf:params:scim:api:messages:2.0:ListResponse:schema" ],
    "name": {
      "formatted": "My Sample Tester III",
      "familyName": "Tester",
      "givenName": "My",
      "middleName": "Sample"
    }
  }),
  headers: { "Authorization": "Bearer " + accessToken },
  contentType: "application/json",
  success: function(){
    // no data returned...
  }
});
```

DELETE

The following is an example call to the Data Broker `/scim/v2/{name}` endpoint that deletes a user entry for ID `9f8a23-47c7be45-0ce5-3105-8ea8-fc3c39c47f91`.

Request

```
DELETE /scim/v2/Users/9f8a23-47c7be45-0ce5-3105-8ea8-fc3c39c47f91
Host: example.com
Authorization: Bearer MF2AAQGGBB1Y1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-LHJGskZwAd3dYWPVERzIAy-VczegSxSR2c5luoiFgSyQFfC_y0kLy15L4iTI
/9f8a23-47c7be45-0ce5-3105-8ea8-fc3c39c47f91==the user's ID
```

Response

```
HTTP/1.1 200 OK
Content-Length: 0
```

jQuery Example

```
$.ajax({
  type: "DELETE",
  url: "https://example.com/scim/v2/Users/"+userId,
  headers: { "Authorization": "Bearer " + accessToken },
  success: function() {

  // no data returned...

  }
});
```

UserInfo Access Example

An OAuth2 client accesses the `/userinfo` endpoint by passing an HTTP `GET` request with an access token parameter to the Data Broker. The response is a JSON object.

Request

The following is a Java Script example call to the Data Broker `/userinfo` endpoint:

```
GET /userinfo
Host: <example.com>
Accept: application/json
Authorization: Bearer MF2AAQGBB1Y1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-LHJGskZwAd3dYWPVERzIAy-VczegSxSR2c5luoiFgSyQFfc_y0kLy15L4iTI
```

Response

The data returned is dependent on the Data Broker configuration and the XACML policies in place.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...
{
  "sub": "9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b",
  "phone_number": "+1 410 030 3103",
  "updated_at": 1409857981,
  "address": {
    "region": "WV",
    "formatted": "Sample User1$30650 Cherry Street$Pensacola, WV 06057",
    "postal_code": "06057",
    "locality": "Pensacola",
    "street_address": "30650 Cherry Street"
  },
  "name": "Sample User1",
  "family_name": "User1",
  "preferred_username": "sampleuser1",
  "given_name": "Sample"
}
```

jQuery Example

```
$.ajax({  
  type: "GET",  
  url: "https://example.com/userinfo",  
  headers: { "Authorization": "Bearer " + accessToken },  
  dataType: "json",  
  success: function(userinfo) {  
    // sample returned data...  
  }  
});
```

Chapter 8: Configuring XACML Policies

XACML policies are the rules that determine what scopes are shared with OAuth2 clients and under what conditions. Policies include the criteria by which access decisions are made using targets, rules, conditions, obligations, and a rule combining algorithm. Several default policies are available, or custom policies can be written.

Topics include:

[XACML Policy Overview](#)

[Policy Structure](#)

[Policy Request Processing Per Endpoint](#)

[Policy Engine Request Context](#)

[Policy Sections and Functions Described](#)

[Configuring the Policy Service](#)

[Policy Information Providers](#)

[Creating Policies](#)

[Creating a Policy Set](#)

[Testing Policies](#)

[Unsupported XACML Features](#)

XACML Policy Overview

Policies determine the scopes that can be accessed by requesting OAuth2 clients through the use of an access token, and the operations on attributes within the scope that are allowed. Policy creation must balance the privacy requirements of the organization with the resource access requirements of the OAuth2 clients. Policies are expressed using the eXtensible access control markup language (XACML) as specified in the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0*, and can contain targets, rules, conditions, and a rule combining algorithm.

XACML policies are evaluated by the Data Broker in response to the following requests made by OAuth2 clients:

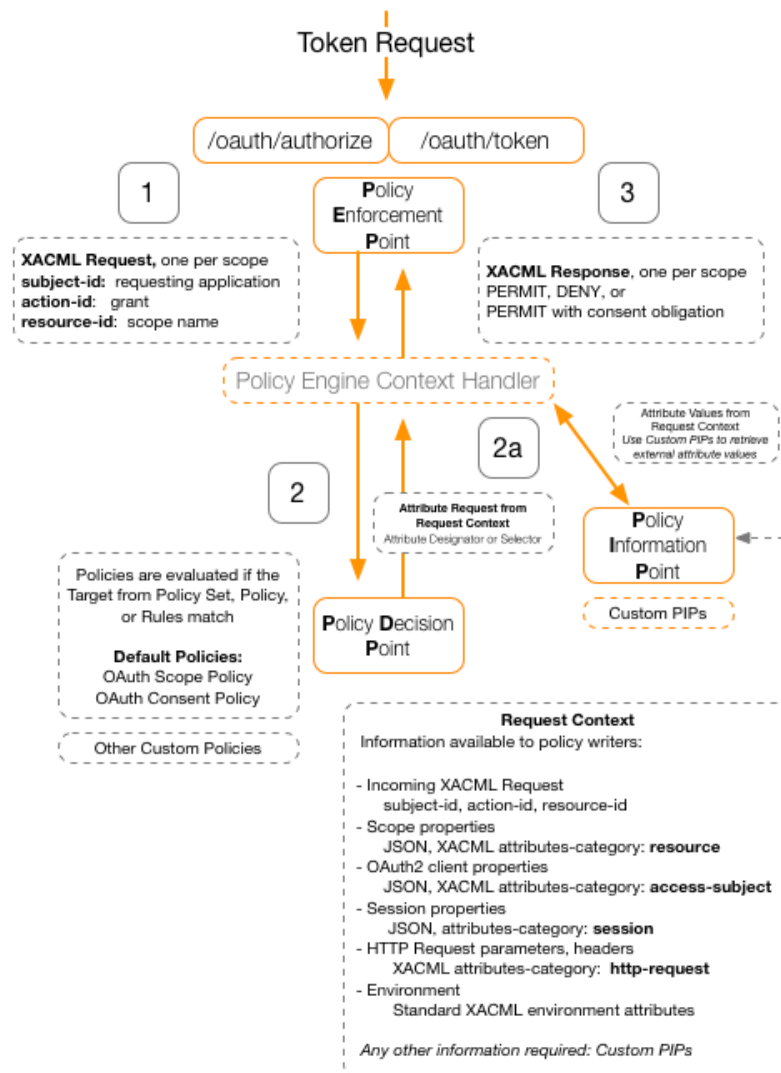
- An authorization/token request to the OAuth2 endpoint.
- A request to the UserInfo endpoint.
- All SCIM requests:
 - Search request
 - Get request
 - Update request
 - Create request
 - Delete request
 - Sub-resource request
- A XACML request to the PDP endpoint.

To create XACML policies that will work as expected, or to create OAuth2 clients that can access data correctly, review the parameters and attributes that will be included in the XACML requests for each of the scenarios provided.

Requesting an Access Token

An OAuth2 client requests an access token, receives the token from the Data Broker or a third-party, and then sends the token to the Data Broker with a set of requested scopes. Each requested scope will generate a policy evaluation, resulting in a `permit` or `deny` to access. Obligations can be used to define conditions for access to each scope, such as requiring user's consent. The token returned to the client after policy evaluation may contain a subset of the requested scopes, or if none of the scopes are granted, no token is returned (the client receives an error response).

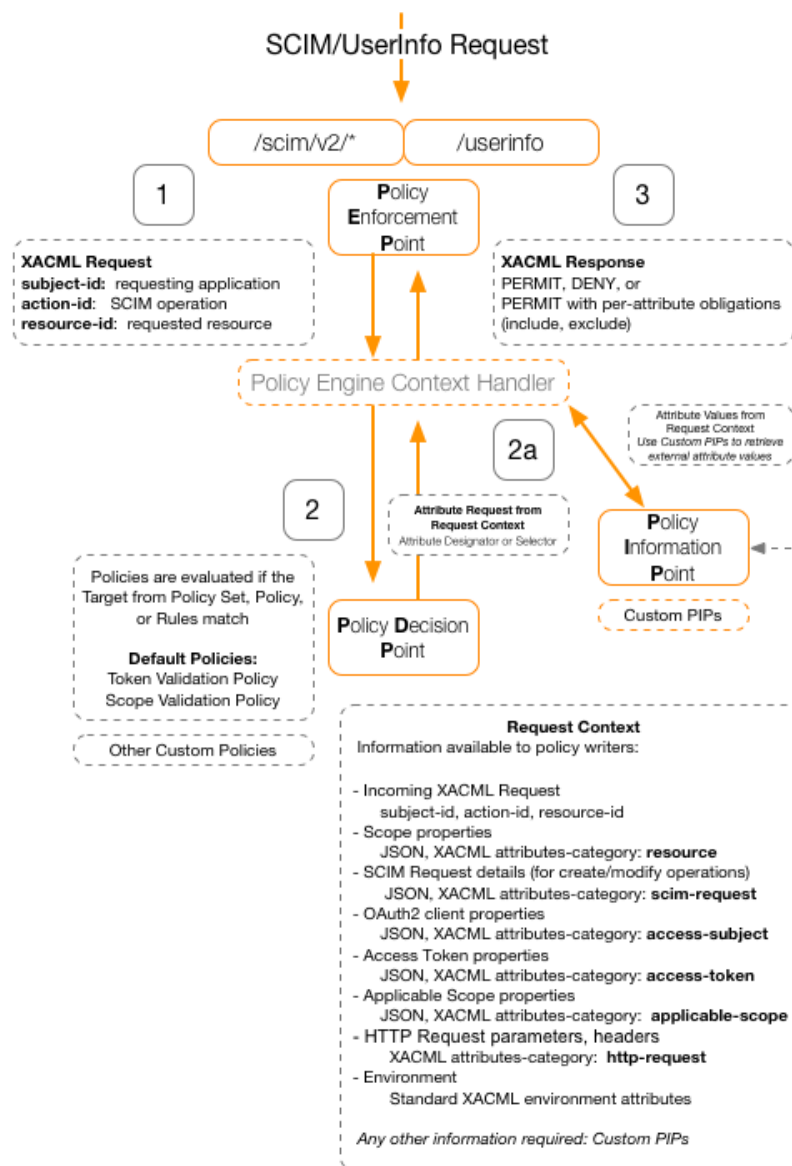
The following illustrates the policy flow for a token request.



Requesting Operations through SCIM or UserInfo

The scopes that policies permit access to are returned in the access token to the OAuth2 client. The token, which represents the privileges granted to the OAuth2 client, may then be sent to either the SCIM or UserInfo endpoint. The Data Broker uses XACML policies again to determine whether the requested operation should be authorized given the scopes granted in the access token. Obligations can again be used to define conditions for limiting access to certain attributes. The requested attributes are returned to the client, and any permitted operation (such as adding or modifying an address) is performed.

The following illustrates the policy flow for a SCIM or UserInfo request.



Policy Structure

For a policy to be evaluated against a request, the request needs to match the values specified in the policy `<Target>` element first. If the target for the request matches the target for the policy, the rules in the policy are evaluated. This occurs for each XACML policy.

Just as there is a target for the policy, there is a target for each rule. For the rule `<Target>` element to be evaluated, a value in the request must match, as defined in the `<Match>` element. If the request matches a value, the rest of the conditions of the rule are evaluated.

Note

If no target is specified for a policy or a rule, the policy or rule is always evaluated.

If the conditions of a rule are satisfied, the result can be either `permit` or `deny` for that single rule. If there are multiple rules in a policy, the rule combining algorithm for the policy determines how the rule evaluation results are combined into a single policy decision.

If there are multiple policies that apply to the request, a policy-combining algorithm determines how the decisions rendered by multiple policies are combined to form a decision by the Data Broker. By default, the combining algorithm for Data Broker policies is `deny-overrides`. This can be changed in the [Policy Service](#) through the Management Console or with the `dsconfig` tool.

Requesting JSON-Formatted Data

The `AttributeSelector` element can be used in a policy to retrieve structured data returned in JSON-format. Differing from the XACML specification, the `Path` references in the `AttributeSelector` are interpreted as JSON paths rather than XPath.

In the following example, an `AttributeSelector` element is used to obtain the region sub-attribute of a user's home address:

```
<AttributeSelector
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
  Path="addresses[type eq home].region"
  DataType="http://www.w3.org/2001/XMLSchema#String"/>
```

Depending on the path specification, an `AttributeSelector` may return multiple nodes from a JSON object, resulting in a XACML "bag" of attribute values. The `DataType` specification of the `AttributeSelector` must specify the type of the node(s) returned. If the nodes returned from the path evaluation are JSON objects rather than a simple data type, then the `AttributeSelector`'s `DataType` must be `http://www.w3.org/2001/XMLSchema#String` and the node value is returned to the Policy Engine as a JSON string.

Using Obligations and Advice

The XACML specification defines an obligation as a specified operation that should be performed by the Policy Enforcement Point (PEP) based on an authorization decision. For example, if certain criteria in a policy rule are met, an obligation for user consent or an additional authorization step may be enforced. Advice is additional information provided to the PEP based on a policy decision, and can be used by the requesting OAuth2 client to determine why [access to a scope or resource was denied](#). The Data Broker provides the following obligation types.

OAuth2 Authorization Requests

Prompt for Consent Obligation – When returned with a permit decision, this obligation indicates that while policies permit the application to have the requested scope, the Data Broker is required to prompt the user for consent before granting a final access token. The obligation may include a `mustGrant` argument. If `true`, the entire OAuth2 request will be rejected if consent is not granted by the end-user. If `false`, and consent is denied by the user, the Identity Provider Service can generate an access token that grants a subset of the initially requested scopes. The default value for `mustGrant` is `true`.

The following is XACML syntax for a sample consent obligation:

```
<ObligationExpressions>
  <ObligationExpression ObligationId="obtain-consent" FulfillOn="Permit">
    <AttributeAssignmentExpression AttributeId="must-grant">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        false
      </AttributeValue>
    </AttributeAssignmentExpression>
  </ObligationExpression>
</ObligationExpressions>
```

SCIM Resource Requests

Exclude Obligation – Specifies an argument that lists the attributes to be excluded from the response. Each attribute must be formatted using SCIM Attribute Notation, such as `urn:scim:schemas:core:2.0:User:userName` for the `userName` attribute of a `User` scope.

Include Obligation – Specifies an argument that lists the attributes to be included in the response. Each attribute must be formatted using SCIM Attribute Notation.

Any attributes not present in either argument list will be excluded from the response. The following example illustrates an exclude obligation that will prevent the `userName` attribute from being returned with a resource:

```
<ObligationExpressions>
  <ObligationExpression ObligationId="exclude-attributes" FulfillOn="Permit">
    <AttributeAssignmentExpression AttributeId="attributeNames">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        userName
      </AttributeValue>
    </AttributeAssignmentExpression>
  </ObligationExpression>
</ObligationExpressions>
```

Policies and Request Processing Per Endpoint

Authorization requests from a client are evaluated by the policy rules configured for the Data Broker. Access to data is granted either at the scope level or at the resource level based on the endpoint through which the request is made. This section describes each type of policy request that may be made by the Data Broker's policy enforcement points. Review [XACML Overview](#) for illustrated processes.

OAuth2 Endpoint Policy Requests

Authorization requests coming through the OAuth2 endpoint are granted if the scopes specified are allowed by configured policies. The authorization endpoint asks for an independent policy decision for each scope requested. If the policy decision for any scope is `deny`, the Identity Provider Service can generate an access token that grants a subset of the initially requested scopes. If all scopes are denied by policy, the entire authorization request is rejected, no access token is issued, and an error response is returned. Policies may return obligations on permit to instruct the Data Broker to perform additional steps before granting the scope.

Once a token is granted, it can be passed to either the SCIM or UserInfo endpoints to retrieve user data. Policies are again evaluated, but at the resource level.

To differentiate authorization requests from resource requests, the Data Broker uses the XACML action type `grant`. This action indicates to XACML policies that the current request is to authorize a scope grant.

Each policy request generated by the authorization endpoint contains the following information.

OAuth2 Authorization Request Attributes		
Attribute ID	Attribute Category	Value
subject-id	access-subject	The client name.
action-id	action	grant.
grant-type	action	The OAuth2 grant type, which is one of <code>authorization_code</code> , <code>implicit</code> , <code>password</code> , or <code>client_credentials</code> .
resource-id	resource	The requested scope name.
<JSON content>	resource	Scope properties.
<JSON content>	access-subject	The OAuth2 client properties.
<JSON content>	session	The session properties including the authenticated user resource.

In addition to these attributes, policies that govern OAuth token requests can obtain, from the [XACML request context](#), details of the underlying HTTP request.

For OAuth2 Scope policy requests originating from the OAuth2 endpoint, details of the requested scope can be accessed from policy using the attribute category `urn:oasis:names:tc:xacml:3.0:attribute-category:resource`. The following example retrieves the list of all operations defined by the scope as a XACML bag:

```
<AttributeSelector
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
  Path="allOperations"
  DataType="http://www.w3.org/2001/XMLSchema#String"/>
```

SCIM Resource Type Policy Evaluation

Each request to the SCIM endpoint explicitly specifies what action is being requested and on what resources. As a REST interface, SCIM uses the HTTP method, query parameters, method body, and URI path to specify request parameters. Policy evaluations generated by the SCIM Resource Type depend on these REST parameters, as well as the supplied OAuth2 bearer token.

All SCIM requests target a specific SCIM Resource Type. For example, a search targeted to `/scim/v2/Users` is executed against the Users SCIM endpoint. An update targeted to `/scim/v2/ConsumerUsers/9f8a23-5f7ec932-55c4-347e-b757-ce74258ea9e6` is executed against a user with ID `9f8a23-5f7ec932-55c4-347e-b757-ce74258ea9e6` in the Users SCIM Resource Type.

SCIM Search Request

A SCIM search request consists of a search filter and an optional specification of which attributes to return from each record that satisfies the filter definition. The SCIM Resource Type against which the search is to be conducted is derived from the relative URL path, such as `/scim/v2/Users`.

The XACML request generated from a SCIM search request contains the following attributes.

SCIM Search Request Attributes

Attribute ID/Content	Attribute Category	Attribute Value
subject-id	access-subject	Name of the requesting OAuth2 client, if it can be retrieved from the OAuth2 access token.
action-id	action	search.
resource-id	resource	Relative URL of the SCIM endpoint, such as <code>Users</code> .
<JSON Content>	access-token	Access token properties.
<JSON Content>	applicable-scope	Applicable scope objects.

After the search is run against the SCIM Resource Type, it generates XACML requests for each record returned in the results to determine whether the requesting client has permission to receive the record's attributes. Each resource and attribute of each record is evaluated independently through a separate policy request to determine if it can be returned. Any resources or individual resource attributes that are denied by policy are omitted from the response. These subsequent policy requests are identical to a SCIM GET request.

Note

The number of search results that can be returned is limited by the SCIM Resource Type's `lookthroughLimit` property, due to the potential cost of checking each response against policy.

SCIM Get Request

The following is contained in the authorization request generated for a SCIM GET request for a known resource.

SCIM GET Request Attributes

Attribute ID/Content	Attribute Category	Value
subject-id	access-subject	Name of the requesting OAuth2 client, if it can be retrieved from the OAuth2 access token.
action-id	action	retrieve.
resource-id	resource	Relative URL of the resource to retrieve, such as <code>Users/12345</code> .
<JSON Content>	resource	SCIM object representation of the requested resource.
<JSON Content>	access-token	Access token properties.
<JSON Content>	applicable-scope	Applicable scope objects.

The SCIM endpoint will perform the following actions based on the result of the XACML policy authorization request:

- If the result is `deny` – The resource is not returned to the client and an error is returned.
- If the result is `permit` – The initial attribute set to be returned to the client is determined. Since multiple policies and/or rules may be consulted to make the permit decision, it's possible that multiple obligations will be returned with the result. See [About Obligations and Advice](#). Include and exclude obligations are processed as follows:
 - All attributes specified in an exclude obligation are removed from the attribute set.
 - If there are include obligations, all attributes that are not specified by an include obligation are removed from the attribute set.
 - If no attributes remain in the attribute set, a 200 success response code is returned but with an empty `resource` object.

These rules for each result type are used for all resources returned from the SCIM endpoint.

SCIM POST Request

The following is contained in the authorization request generated for a SCIM POST request.

SCIM POST Request Attributes		
Attribute ID	Attribute Category	Value
<code>subject-id</code>	<code>access-subject</code>	The client application name.
<code>action-id</code>	<code>action</code>	<code>create</code> .
<code>resource-id</code>	<code>resource</code>	Relative URL of the SCIM Resource Type to be created, such as <code>Users</code> .
<code><JSON Content></code>	<code>scim-request</code>	SCIM request body of the resource to be created.
<code><JSON Content></code>	<code>access-token</code>	Access token properties.
<code><JSON Content></code>	<code>applicable-scope</code>	Applicable scope objects.

If the POST operation is permitted, the new resource is created and the new object is returned to the client. After the POST is complete, a second policy request is issued to determine which attributes of the updated record the client can receive in the response.

SCIM PATCH and PUT Requests

PUT requests are internally converted into a PATCH operation, which is why they are handled the same way by policy. The following is contained in the authorization request generated for a SCIM PATCH or PUT request for a known resource.

SCIM PATCH Request Attributes		
Attribute ID	Attribute Category	Value
<code>subject-id</code>	<code>access-subject</code>	The client application name.
<code>action-id</code>	<code>action</code>	<code>modify</code> .

SCIM PATCH Request Attributes

Attribute ID	Attribute Category	Value
resource-id	resource	Relative URL of the resource to be modified, such as <code>Users/12345</code> .
<JSON Content>	scim-request	The normalized SCIM PATCH request body.
<JSON Content>	access-token	Access token properties.
<JSON Content>	applicable-scope	Applicable scope objects.

If the PATCH or PUT operation is permitted, the resource is updated and returned to the client. The updated resource is then subject to the same read criteria in a GET request.

SCIM Delete Request

The following is contained in the authorization request generated for a SCIM DELETE request for a known resource.

SCIM DELETE Request Attributes

Attribute ID	Attribute Category	Value
subject-id	access-subject	The client application name.
action-id	action	<code>delete</code> .
resource-id	resource	Relative URL of the resource to be deleted, such as <code>Users/12345</code>
<JSON Content>	access-token	Access token properties.
<JSON Content>	applicable-scope	Applicable scope objects.

SCIM Sub-Resource Operation Policy Evaluation

The Data Broker can return account status, password restrictions, consent records, consent history, and external identity provider information for authenticated identities. Policy evaluation for requests that include account, consent, or external identity operations require that the requested [OAuth2 scopes](#) include the desired action, and the request is made to the correct sub-resource endpoint.

Note

Account, password, and one-time token delivery operations depend on the Data Store's Password Policy State Extended Operation configuration. See the *UnboundID Data Store Administration Guide* for configuration details.

Sub-resource endpoints include:

- `/scim/v2/<scim-resource-type>/<id>/account` – Processes requests to retrieve or replace an account's state.
- `/scim/v2/<scim-resource-type>/<id>/consents` – Processes requests to retrieve or revoke a user's consent to access resources.

- `/scim/v2/<scim-resource-type>/<id>/externalIdentities` – Processes requests to link, unlink, or retrieve account information from a configured external identity provider.
- `/scim/v2/<scim-resource-type>/<id>/password` – Processes requests to reset an account password.
- `/scim/v2/<scim-resource-type>/<id>/passwordQualityRequirements` – Processes requests to retrieve the configured password requirements as defined in the Data Store's default password policy.

Account Operations

The following are included in the authorization request generated for an account operation request for a known resource.

SCIM Account Operation Request Attributes

Attribute ID	Attribute Category	Value
subject-id	access-subject	The OAuth2 client name, if it can be obtained from the access token.
resource-id	resource	Relative URL of the parent resource, such as <code>/Users/12345</code> .
action-id	action	The account action to perform as listed in the requested scope .

Consent Operations

The following are included in the authorization request generated for a consent operation request for a known resource.

SCIM Consent Operation Request Attributes

Attribute ID	Attribute Category	Value
subject-id	access-subject	The OAuth2 client name, if it can be obtained from the access token.
resource-id	resource	Relative URL of the parent resource, such as <code>/Users/12345</code> .
action-id	action	The consent action to perform as listed in the requested scope .

External Identity Provider Operations

The following are included in the authorization request generated for a consent operation request for a known resource.

SCIM External Identity Provider Operation Request Attributes

Attribute ID	Attribute Category	Value
subject-id	access-subject	The OAuth2 client name, if it can be obtained from the access token.
resource-id	resource	Relative URL of the parent resource, such as <code>/Users/12345</code> .
action-id	action	The external identity account action to perform as listed in the requested scope .

UserInfo Endpoint Policy Evaluation

The UserInfo endpoint interaction with the policy engine is identical to a SCIM GET operation against the `/Me` endpoint.

Policy Decision Point (PDP) Endpoint

The PDP endpoint enables an external Policy Enforcement Point (PEP) to generate XACML requests and send them directly to the Data Broker for evaluation. The request is passed directly to the policy engine. The request can contain any standard XACML attributes, Data Broker custom attributes, or other attributes that may be required by custom policies. This endpoint requires that the client authenticate using HTTP basic authentication.

Policy Engine Request Context

The XACML policy request context contains the information that is available to the policy engine to make a decision. A request for authorization (OAuth2) will provide information that helps the policy engine determine whether or not an OAuth2 client should be granted or denied access to a scope. A request for resources (SCIM or UserInfo) will provide information that will help determine if the operations on attributes in the requested scopes can be performed.

The request context contains attributes directly passed by a client when making an authorization request to the policy engine. It is supplemented with additional attributes and JSON objects that are retrieved from the [attribute categories](#). In order to make a policy decision, policies can reference any attribute or JSON object from the request context.

XACML Attribute Categories

All references from policy to objects that can be obtained from the request context are first identified by their XACML attribute category.

- `urn:oasis:names:tc:xacml:3.0:attribute-category:resource` – This standard XACML category definition is always used to reference the object to which authorization is being requested. With a SCIM request, this is a SCIM resource whose type is determined by the SCIM request path. With an OAuth2 request, it will reference a Scope object. In either case, the request context exposes the resource as a JSON object that

policies can access using `AttributeSelector` elements. For the `consent`, `account` and `external-identity` sub-resources, the JSON content will be that of the parent user resource. See [Resource Properties](#) for details.

- `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject` – This standard XACML category definition can be used in an `AttributeSelector` to obtain attributes of the OAuth2 client, on whose behalf the policy request has been made. See [OAuth2 Client Properties](#) for details.
- `urn:unboundid:names:2.0:attribute-category:access-token` – This custom category provides access to properties of the OAuth2 access token that has been used to make the current request. It exposes the access token as a JSON object that can be accessed using `AttributeSelector` elements. See [Processing Access Tokens](#) for details.
- `urn:unboundid:names:2.0:attribute-category:http-request` – This custom category provides access to properties of the incoming HTTP request that triggered the policy request. HTTP headers and query parameters are available through UnboundID-defined `AttributeDescriptors`. See [HTTP Request Properties](#) for details.
- `urn:unboundid:names:2.0:attribute-category:scim-request` – This custom category is populated by the Data Broker SCIM endpoint and contains the JSON request body of the SCIM request that triggered policy evaluation. Policies that target SCIM requests can retrieve details of the incoming request using `AttributeSelector` elements. The content from this attribute category is in standard SCIM 2.0 format. See [SCIM Request Properties](#) for details.
- `urn:unboundid:names:2.0:applicable-scope` – This custom category is populated with the scopes from the access token that are applicable to authorize a resource request. See [Applicable Scopes](#) for details.
- `urn:unboundid:names:2.0:session` – This custom category provides access to properties of the current Data Broker session, if one exists. See [Session Properties](#) for details.

Other attribute categories can be defined by custom PIPs.

Standard XACML Attribute Use

The following request attributes are specified by the XACML specification. Unless otherwise specified, these are always available in the Data Broker's XACML request context.

Per the XACML specification, any attribute retrieved from the request context with an `AttributeDescriptor` element will be a 'bag' (XACML term) of attribute values. Where the attribute has a single value, the value can be extracted from the bag using a `type-one-and-only` XACML function (see section A.3.10 of the XACML specification, "Bag functions").

Standard XACML Attributes

Attribute URN	Attribute Category	XACML Data Type	Description
urn:oasis:names:tc:xacml:1.0:subject:subject-id	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	string	Contains the name of the OAuth2 client that is submitting a policy request, as specified when the client is registered with the Data Broker.
urn:oasis:names:tc:xacml:3.0:subject:authnlocality:ip-address	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	ipAddress	Contains the originating IP address of the client's authorization request. The availability and accuracy of this attribute is dependent upon the deployed Data Broker's network environment. When available, the value is retrieved from the XFORWARDED_FOR header of the client's HTTP request. If that header is not available, the IP address returned may be that of the last proxy to send the request.
urn:oasis:names:tc:xacml:1.0:resource:resource-id	urn:oasis:names:tc:xacml:3.0:attribute-category:resource	anyURI	Contains the URN of the resource being requested.
urn:oasis:names:tc:xacml:1.0:action:action-id	urn:oasis:names:tc:xacml:3.0:attribute-category:action	string	Contains the name of the action being requested. The action-id will be <code>grant</code> for OAuth2 requests, and will correspond to one of the scope operations .
urn:oasis:names:tc:xacml:1.0:environment:current-time	urn:oasis:names:tc:xacml:3.0:attribute-category:environment	time	The time at which the Data Broker began processing the current authorization request.
urn:oasis:names:tc:xacml:1.0:environment:current-date	urn:oasis:names:tc:xacml:3.0:attribute-category:environment	date	The date on which the current authorization request is being processed.
urn:oasis:names:tc:xacml:1.0:environment:current-dateTime	urn:oasis:names:tc:xacml:3.0:attribute-category:environment	dateTime	The date and time at which the Data Broker began processing the current authorization request.

Custom XACML Function

There is a single custom function implemented by the Data Broker. All other functions supported by the policy engine are XACML standard functions.

The `urn:unboundid:names:2.0:function:scimAttribute-subset` function is similar to the standard XACML string-subset function, except that the arguments are bags of SCIM attribute names using SCIM attribute notation as described in the SCIM specification. The custom function comprehends wildcard attribute specifications as supported in the `resourceAttributes` property of a [Data Broker OAuth2 scope](#).

For example, if the second set passed to this function contains the string `urn:mySchema:*`, and the first set contains `urn:mySchema:myAttribute`, the function may still return TRUE (the first set is considered to be a subset of the second).

Resource Properties

SCIM Resource Type resources are exposed as JSON objects that can be accessed from policy using `AttributeSelector` elements. By default, the only attribute that can be accessed using an `AttributeDesignator` is `resource-id`. For user-defined resources such as Users, the format of the JSON object is determined by the structure of the underlying resource and the mappings defined for its SCIM Resource Type. Depending on the type of request, the contents of the resource category may be either a SCIM Resource or a scope.

Scope Properties

When an OAuth2 client makes an authorization request using the standard OAuth2 endpoints, the resource category content is a scope object. Based on the OAuth2 client's configuration, configured XACML policies, and consent requirements, the Data Broker will decide which scopes to grant in the access token.

By default, the Data Broker only authorizes the scope. The OAuth2 client bearing the granted token cannot use it to obtain any attributes or claims. See [OAuth2 Scopes](#) for details about creating scopes.

SCIM Resource Properties

When an OAuth2 client makes a request through the SCIM or Userinfo endpoints, the resource category content is a SCIM Resource. For example, this `AttributeSelector` will retrieve the `region` sub-attribute of a user's home address within the requested User resource.

```
<AttributeSelector
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
  Path="addresses[type eq 'home'].region"
  DataType="http://www.w3.org/2001/XMLSchema#String"/>
```

Accessing Referenced SCIM Resource Attributes

The Data Broker supports referenced attributes, as described in the SCIM 2.0 Core Schema specification. When the reference is to another SCIM object, a policy can be used to follow the reference link and retrieve attributes of the referenced object using an `AttributeSelector`. The policy must use the `ContextSelectorId` element of the `AttributeSelector` as the path to the reference attribute. The `Path` element is then interpreted as the JSON path into the referenced object.

In the following example, a Credit Cards SCIM Resource Type contains registered credit card objects for all users, and a User SCIM Resource Type that has a multivalued `paymentMethods` attribute that contains a list of payment object references, some of which are credit cards. The following `AttributeSelector` will retrieve a XACML bag containing the expiration dates for all credit cards registered to the user.

```
<AttributeSelector
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
  Path="expirationDate"
  ContextSelectorId="paymentMethods[paymentType eq &quot;credit&quot;].$.ref"
  DataType="http://www.w3.org/2001/XMLSchema#date"/>
```

The value of the `ContextSelectorId` must resolve to (one or more) relative URIs whose value is of the form `CreditCards/<Id>`, where the ID is a unique credit card object ID.

Note

Policies are not able to resolve SCIM reference attributes whose value is an external or absolute URI.

OAuth2 Client Properties

Properties of the requesting OAuth2 client are exposed as a JSON object under the XACML attribute category `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`.

By default, the only attribute that can be accessed using an `AttributeDesignator` is `subject-id`. Other attributes, defined when the OAuth2 client is added to the Data Broker, may be accessed using an `AttributeSelector`, including the following properties.

OAuth2 Client Properties

Property	Data Type	Description
<code>grantType</code>	Multivalued string.	A list of OAuth2 grant types that this client is authorized to use.
<code>scope</code>	JSON Object array.	The scopes associated with the OAuth2 client.
<code>clientId</code>	String.	The OAuth client ID.
<code>name</code>	String.	The OAuth client name.
<code>tags</code>	Multivalued string.	A list of tags associated with this OAuth2 client.

Scope Properties

The default OAuth2 Scope policy allows scope operations as long as one of the scopes granted in the access token allows the operation. Access to attributes allowed per operation is the union of all `resourceAttributes` defined in [Authenticated Identity](#) or [Resource](#) scopes that allow that operation.

In order for operations to be allowed on resources, the XACML policies that process the requests must allow the operations requested in the scope. The following scope properties can be evaluated by policies.

Scope Properties		
Property	Data Type	Description
tokenName	String.	The scope name as presented in an OAuth2 request.
type	String.	The scope type, which is <code>authenticated-identity</code> for authenticated identity scopes, <code>resource</code> for resource scopes, or <code>oauth2</code> for a generic scope.
tags	String. Multivalued.	A list of Tags associated with a scope that can be examined by XACML policies.
scimResourceType	Aggregation.	If a <code>resource</code> scope, the SCIM Resource Type that can be accessed.
resourceOperations	Multivalued list. Optional.	<p>Operations can include:</p> <ul style="list-style-type: none"> • <code>retrieve (GET)</code> from endpoint <code>/scim/v2/<id></code> • <code>modify (PATCH or PUT)</code> to endpoint <code>/scim/v2/<id></code>
accountOperations	Multivalued list. Optional.	<p>Operations can include:</p> <ul style="list-style-type: none"> • <code>reset-password (PUT)</code> to endpoint <code>/scim/v2/<id>/password</code> • <code>retrieve-password-quality-requirements (GET)</code> from endpoint <code>/scim/v2/<id>/passwordQualityRequirements</code> • <code>retrieve-account-state (GET)</code> from endpoint <code>/scim/v2/<id>/account</code> • <code>replace-account-state (PUT)</code> to endpoint <code>/scim/v2/<id>/account</code>
consentOperations	Multivalued list. Optional.	<p>Operations can include:</p> <ul style="list-style-type: none"> • <code>retrieve-consent (GET)</code> from endpoint <code>/scim/v2/<id>/consents</code> or <code>/scim/v2/<id>/consents/<id></code> • <code>revoke-consent (DELETE)</code> from endpoint <code>/scim/v2/<id>/consents/<id></code> • <code>retrieve-consent-history (GET)</code> from endpoint <code>/scim/v2/<id>/consentHistory/<id></code>
externalIdentity Operations	Multivalued list. Optional.	<p>Operations can include:</p> <ul style="list-style-type: none"> • <code>retrieve-external-identity (GET)</code> from endpoint <code>/scim/v2/<id>/externalIdentities</code> or <code>/scim/v2/<id>/externalIdentities/<id></code> <p>This will expose access tokens from the identity provider.</p> <ul style="list-style-type: none"> • <code>unlink-external-identity (DELETE)</code> from

Scope Properties		
Property	Data Type	Description
		endpoint /scim/v2/<id>/externalIdentities/<id>
allOperations	Multivalued list.	A computed value containing the union of all operations defined by resourceOperations, accountOperations, consentOperations, and externalIdentityOperations.
resourceAttributes	Multivalued string.	A list of one or more SCIM attributes of the authenticated identity for which this scope allows access. The type of access is determined by the operation properties retrieve, replace, and modify. A wildcard value of * can be used for all attributes. A schema-specific wildcard value of the form urn:<schemaName>:* can be used to represent all attributes of a single schema namespace. Access to attributes allowed per operation is the union of all resourceAttributes allowed in the scope.

HTTP Request Properties

The XACML request context exposes some properties of the HTTP request. The HTTP request will be either an OAuth2 request, a SCIM request, a UserInfo request, or a PDP request. All access to the HTTP request is through the XACML attribute category

urn:unboundid:names:2.0:attribute-category:http-request.

HTTP header values can be obtained using an AttributeDesignator with AttributeId of the form urn:unboundid:names:2.0:http-request:header:<header-name>, where header-name is the name of the header requested. The following example retrieves the value of the Cookie header:

```
<AttributeDesignator Category="urn:unboundid:names:2.0:attribute-category:http-request"
  AttributeId="urn:unboundid:names:2.0:httpHeader:Cookie"
  DataType="http://www.w3.org/2001/XMLSchema#string"
  MustBePresent="false"/>
```

HTTP query parameters can be obtained using an AttributeDesignator with AttributeId of the form urn:unboundid:names:2.0:httpQueryParam:<parameter-name>, where parameter-name is the name of the query attribute requested. The following example retrieves the value of the query parameter with name channel:

```
<AttributeDesignator Category="urn:unboundid:names:2.0:attribute-category:http-request"
  AttributeId="urn:unboundid:names:2.0:httpQueryParam:channel"
  DataType="http://www.w3.org/2001/XMLSchema#string"
  MustBePresent="false"/>
```

SCIM Request Properties

For policy evaluation of SCIM requests, the HTTP message body, if one exists, is available as the content of the scim-request attribute category. The content type of a SCIM request is always JSON, so the request body can be accessed using an AttributeSelector with a JSON path. For convenience, the attribute with ID urn:unboundid:names:2.0:impacted-

`attributes` is also available. This attribute is computed by the policy engine and returns a XACML bag of attribute names in SCIM attribute notation. It returns only the attributes that can be created, modified, or deleted as a result of a SCIM POST, PUT, or PATCH request. See the SCIM 2.0 specification for more details.

The following example retrieves all impacted attributes from the current SCIM request:

```
<AttributeDesignator
  Category="urn:unboundid:names:2.0:attribute-category:scim-request"
  AttributeId="urn:unboundid:names:2.0:impacted-attributes"
  DataType="http://www.w3.org/2001/XMLSchema#string">
```

Applicable Scopes

An OAuth2 access token presented by an OAuth2 client to the Data Broker can contain many scopes, only some of which are applicable to the current request. The Data Broker's PIP exposes the applicable scopes under the XACML attribute category `urn:unboundid:names:2.0:attribute-category:applicable-scope`. This category contains a list of JSON scope objects, described in [OAuth2 Scopes](#), for those scopes granted by the access token that meet the following criteria:

- The current request's `action-id` is contained in one of the scope's operations properties.
- The type of resource requested matches the type of resource to which the scope grants access. For Authenticated Identity scopes, they are only applicable to requests in which the resource requested is the access token owner.
- Generic OAuth2 scopes are always included since their meaning is not defined by the Data Broker.

The following example retrieves all attributes that are granted access by all applicable scopes of the access token:

```
<AttributeSelector
  Category="urn:unboundid:names:2.0:attribute-category:applicable-scope"
  Path="scope.resourceAttributes"
  DataType="http://www.w3.org/2001/XMLSchema#String"/>
```

Session Properties

An authenticated identity must be established to obtain an OAuth2 access token using all OAuth2 grant types, except for Client Credentials. During policy evaluation of an OAuth2 access token grant request, the XACML attributes category `urn:unboundid:names:2.0:attribute-category:session` contains JSON content describing the currently authenticated user. The following properties are available in this attribute category.

Session Properties

Property	Data Type	Description
<code>sub</code>	String	Relative SCIM path to the authenticated user resource, such as <code>Users/123456789</code> .
<code>subResource</code>	JSON object	The SCIM resource object for the token owner.

Access Token Properties

The Data Broker's Access Token Policy Information Provider (PIP) exposes Data Broker-generated access tokens as JSON objects under the XACML attribute category `urn:unboundid:names:2.0:attribute-category:access-token`. The properties of a Data Broker access token adhere to the JSON Web Token specification, with some Broker-specific extensions.

The following properties are available in the access token category.

Access Token Properties		
Property	Data Type	Description
<code>active</code>	Boolean. Required.	<code>True</code> if the token is valid, <code>false</code> if token is invalid or has expired.
<code>sub</code>	String. Required.	The unique identifier for the token owner. For user tokens, this will be the relative SCIM path to the user resource, such as <code>Users/123456789</code> . For Client Credentials (<code>app</code>) tokens, this is populated with the OAuth2 client name.
<code>subResource</code>	JSON object.	The SCIM resource object for the token owner. This is not present for <code>app</code> tokens.
<code>typ</code>	String.	The type of token, either <code>user</code> or <code>app</code> .
<code>scope</code>	Multivalued string.	A list of scope names granted by this token.
<code>app</code>	String.	The name of the OAuth2 client for which this token was created. For application tokens, this value will be equal to <code>sub</code> .
<code>iat</code>	DateTime.	The date and time at which the token was created.
<code>exp</code>	DateTime.	The date and time at which the token will expire.
<code>jti</code>	String.	The unique token identifier.

Note

For OAuth2 grant requests, there is no access token available in the request context, since the request is to obtain a new token.

The following example retrieves the entitlements of the access token owner:

```
<AttributeSelector
  Category="urn:unboundid:names:2.0:attribute-category:access-token"
  Path="sub.entitlements"
  DataType="http://www.w3.org/2001/XMLSchema#String"/>
```

Policy Sections and Functions Described

The following is the Scope Validation policy, installed with the Data Broker. This policy is applied to all incoming SCIM requests. Each section and its function is described to show how a policy is constructed. Use this to determine how to create new policies or modify existing ones.

The Scope Validation Policy

```

1<?xml version="1.0" encoding="UTF-8"?>
2<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
3    PolicyId="urn:unboundid:policy:ScopeValidationPolicy"
4    Version="1"
5    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
overrides">
6    <Description>
7        Authorizes requests based on the scopes granted by the provided access token.
8    </Description>
9    <Target/>
10    <!-- The XACML action-id (requested operation) -->
11    <VariableDefinition VariableId="action-id">
12        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
13            <AttributeDesignator
14                MustBePresent="true"
15                Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
16                AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
17                DataType="http://www.w3.org/2001/XMLSchema#string"/>
18        </Apply>
19    </VariableDefinition>
20    <!-- whether the granted scope(s) permit access to all resource attributes -->
21    <VariableDefinition VariableId="allAttributesAllowed">
22        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
23            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
24                *
25            </AttributeValue>
26            <AttributeSelector Category="urn:unboundid:names:2.0:attribute-
category:applicable-scope"
27                Path="scope.resourceAttributes"
28                DataType="http://www.w3.org/2001/XMLSchema#string"
29                MustBePresent="false"/>
30        </Apply>
31    </VariableDefinition>
32    <Rule RuleId="urn:unboundid:rule:ApplicableScope" Effect="Deny">
33        <Description>
34            Deny access if the requested action is not allowed by any scope in the access
35            token.
36        </Description>
37        <Condition>
38            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
39                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
40                    <VariableReference VariableId="action-id"/>
41                    <AttributeSelector Category="urn:unboundid:names:2.0:attribute-
category:applicable-scope"
42                        Path="scope.allOperations"
43                        DataType="http://www.w3.org/2001/XMLSchema#string"
44                        MustBePresent="false"/>
45                </Apply>
46            </Apply>
47        </Condition>
48        <AdviceExpressions>
49            <AdviceExpression AdviceId="request-denied-reason" AppliesTo="Deny">
50                <AttributeAssignmentExpression AttributeId="error">
51                    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
52                        insufficient_scope

```

Chapter 8: Configuring XACML Policies

```
53         </AttributeValue>
54     </AttributeAssignmentExpression>
55     <AttributeAssignmentExpression AttributeId="error-description">
56         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
57             Requested operation not allowed by the granted OAuth2 scopes.
58         </AttributeValue>
59     </AttributeAssignmentExpression>
60 </AdviceExpression>
61 </AdviceExpressions>
62 </Rule>
63 <Rule RuleId="urn:unboundid:rule:AllowOnlyScopedAttributes" Effect="Deny">
64     <Description>
65         For create and modify operations, deny if the request impacts attributes
66         that are not allowed by the applicable scopes.
67     </Description>
68     <Condition>
69         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
70             <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
71                 <VariableReference VariableId="action-id"/>
72                 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
73                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
74                         create
75                     </AttributeValue>
76                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
77                         modify
78                     </AttributeValue>
79                 </Apply>
80             </Apply>
81             <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
82                 <VariableReference VariableId="allAttributesAllowed"/>
83             </Apply>
84             <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
85                 <Apply FunctionId="urn:unboundid:names:2.0:function:scimAttribute-subset">
86                     <AttributeDesignator Category="urn:unboundid:names:2.0:attribute-
category:scim-request"
87                                     AttributeId="urn:unboundid:names:2.0:impacted-
attributes"
88                                     DataType="http://www.w3.org/2001/XMLSchema#string"
89                                     MustBePresent="false"/>
90                     <AttributeSelector Category="urn:unboundid:names:2.0:attribute-
category:applicable-scope"
91                                     Path="scope.resourceAttributes"
92                                     DataType="http://www.w3.org/2001/XMLSchema#string"
93                                     MustBePresent="false"/>
94                 </Apply>
95             </Apply>
96         </Apply>
97     </Condition>
98     <AdviceExpressions>
99         <AdviceExpression AdviceId="request-denied-reason" AppliesTo="Deny">
100             <AttributeAssignmentExpression AttributeId="error">
101                 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
102                     insufficient_scope
103                 </AttributeValue>
104             </AttributeAssignmentExpression>
105             <AttributeAssignmentExpression AttributeId="error-description">
106                 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
```

```

107         Request includes attributes not allowed by the granted OAuth2 scopes.
108     </AttributeValue>
109 </AttributeAssignmentExpression>
110 </AdviceExpression>
111 </AdviceExpressions>
112 </Rule>
113 <Rule RuleId="urn:unboundid:rule:IncludeOnlyScopedAttributes" Effect="Permit">
114     <Description>
115         For retrieve requests, limit the attributes returned to those specifically
116         allowed by the applicable scopes.
117     </Description>
118     <Condition>
119         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
120             <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
121                 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
122                     retrieve
123                 </AttributeValue>
124                 <VariableReference VariableId="action-id"/>
125             </Apply>
126             <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
127                 <VariableReference VariableId="allAttributesAllowed"/>
128             </Apply>
129         </Apply>
130     </Condition>
131     <ObligationExpressions>
132         <ObligationExpression ObligationId="include-attributes" FulfillOn="Permit">
133             <AttributeAssignmentExpression AttributeId="attributeNames">
134                 <AttributeSelector Category="urn:unboundid:names:2.0:attribute-
category:applicable-scope"
135                     Path="scope.resourceAttributes"
136                     DataType="http://www.w3.org/2001/XMLSchema#string"
137                     MustBePresent="false"/>
138             </AttributeAssignmentExpression>
139         </ObligationExpression>
140     </ObligationExpressions>
141 </Rule>
142</Policy>

```

Section Descriptions

Sections are described by line numbers.

- [3] The `PolicyId` specification, must be unique among all policies installed in the Data Broker.
- [5] The `deny-overrides` combining algorithm indicates that if any rule results in a `deny`, then the result of the policy will be `deny`.
- [6-8] The description is displayed when this policy is viewed in the Management Console. This policy authorizes requests based on the scopes granted by the provided access token.
- [9] The `Target` specification for the policy. This is empty because the policy is intended to be used inside a policy set that will set the target.

Chapter 8: Configuring XACML Policies

- [10-19] Since the `action-id` is used multiple times in the policy, it is defined as a XACML variable.
- [21] This boolean XACML variable will be `true` if the access token allows access to all attributes of the requested resource.
- [24-27] The `AttributeSelector` returns a XACML bag containing the value of the `resourceAttributes` property for each scope granted by the access token. If any value in the bag contains a wildcard (*), that indicates that all attributes are accessible.
- [32-47] The first rule in the policy denies the request if the requested action is not allowed by any scope in the access token.
- [48-61] Provides an `AdviceExpression` that an error should be returned with the reason that access was denied, "Requested operation not allowed by the granted OAuth2 scopes." See [Troubleshooting Denied Access](#).
- [62-112] This rule only applies to `create` and `modify` requests, and does not allow any request that impacts attributes that are not included in the access token's scopes.
- [69] The rule's condition consists of three clauses that all must be true (with an AND condition).
- [70-80] This clause checks that the action requested is one of `create` or `modify`.
- [81-83] This clause checks that the access token does not allow access to all attributes (with a wild-card).
- [84-96] This clause ensures that the attributes impacted by the incoming request are a subset of the attributes granted by the access token's scopes.
- [98-111] Provides an `AdviceExpression` that an error should be returned with the reason that operations were denied, "Request includes attributes not allowed by the granted OAuth2 scopes."
- [113-141] This rule applies only to retrieve requests, and uses an obligation to limit the attributes returned in the response.
- [132] The obligation is of type `include-attributes`. It is only applied if the result of the rule is `Permit`.
- [133- 138] The obligation argument contains the names of all attributes that may be returned. The `AttributeSelector` returns the union of attributes allowed by each applicable scope.

Configuring the Policy Service

XACML policies are managed by the Policy Service. The default conditions of the Policy Service can be viewed and changed with the `dsconfig` tool, or through the Management Console **Authorization and Policies -> XACML Policy Service**.

The one property that can be changed is the **combining-algorithm**, which determines how decisions are made if multiple policies or policy sets are applied to a request for resources. The default for the Policy Service is `deny-overrides`, which specifies that a "deny" decision from a policy should take priority over a "permit" decision. The Data Broker also supports `permit-overrides`, `deny-unless-permit`, and `permit-unless-deny`. See the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0. August 2010* for details about each combining algorithm.

Policy Information Providers

Policy Information Providers are used to retrieve XACML attribute(s) from the Policy Information Point (PIP) during policy evaluation. See [Standard XACML Attribute Use](#) and [Custom XACML Attribute Use](#) for information about these attributes. The Data Broker provides the following Policy Information Providers:

BuiltIn Policy Information Provider – Resolves XACML attributes that are implemented by the Data Broker.

SCIM Request Policy Information Provider – Resolves XACML attributes whose value can be retrieved from an incoming SCIM request.

SCIM Resource Type Policy Information Provider – Resolves XACML attributes whose value can be retrieved from a SCIM Resource Type configured on this Data Broker instance.

Token Policy Information Provider – Resolves XACML attributes whose value can be retrieved from an OAuth2 access token generated by this Data Broker instance.

Note

If XACML policies must process requests that rely on [third-party tokens](#) or data, a custom PIP must be created with the UnboundID Server SDK.

PIP Evaluation Order

When multiple PIPs are defined, the evaluation order determines the correct provider to verify a specified XACML attribute. Each PIP must have a unique evaluation value defined within a Data Broker instance. PIPs with a smaller value are evaluated first to determine if they match a XACML attribute ID.

Creating XACML Policies

The Management Console, **Authorization and Policies -> XACML Policies**, or the `dsconfig` tool can be used to create and manage XACML policies. Policies that are written or imported must be syntactically correct and:

- Contain all required policy elements required by XACML 3.0.
- Not contain optional elements that are not supported by the Data Broker.
- Pass XACML function checks for the correct number and type of parameters.

If any of these criteria are not met, the create or import fails.

Several policies are available by default and can be used as templates or adjusted to fit specific requirements:

OAuth2 Policy Set – A container for policies that apply to OAuth2 authorization requests. Each policy referenced in this set is evaluated in order. If a policy returns a `deny`, policies listed after that are not evaluated.

OAuth2 Consent – Determines whether consent is required before granting the requested scope to an OAuth2 client. This policy will deny scopes of type `OAuth2` if the grant type is Client Credentials, since user consent cannot be obtained with this grant type.

OAuth2 Scope – Determines whether a client making an OAuth2 authorization request should be granted a requested scope. If any rule in the policy results in `deny`, the policy will deny the scope. The OAuth2 client making the request must be configured in the Data Broker to request the scope.

SCIM Resource Policy Set – A container for policies that authorize requests for protected resources, including SCIM and UserInfo requests.

Scope Validation – Authorizes SCIM requests based on the scopes granted by the access token provided. The scope must also be configured to enable a requested action. See [OAuth2 Scopes](#) for details.

Token Validation – Denies all SCIM resource requests that do not contain a valid access token.

Creating a Policy Set

A policy set is an ordered collection of policies that work together to perform a policy task. The policy set is a XACML-defined entity. The Data Broker evaluates policy sets the same way it evaluates policies.

Creation of a policy set is the same as that of a policy. A policy set must be created from individual policies that have been configured in the Data Broker.

Note

Policies that are part of a policy set should be disabled in the Data Broker, once the policy set is enabled. This will prevent policies from being evaluated twice.

Testing Policies

Policies can be tested by running request scenarios through the API Explorer to ensure that they work as designed before deploying in production. The API Explorer can be used to create an authorization request, specify the OAuth2 client that will request access to a user's

resources, the resources to access, and additional information from the user's entry to assist in processing the request. See [About Data Access Requests](#) for an overview of the request components.

Access the API Explorer from the Documentation Index page, `<server-root>/docs/index.html`, or the server's HTTPS endpoint `https://<host>:<http-port>/explorer`.

Troubleshooting Policies with Traces

Policy decisions are frequently the result of a complex series of logical steps. Identifying the reason why a particular request is getting an unexpected result can be difficult. The Data Broker can generate a trace of any policy decision, and log traces with in the File Based Trace Log Publisher, or with XACML Policy Trace Filters created with `dsconfig` or through the Management Console, **Authorization and Policies -> XACML Policy Trace Filters**.

Note

Policy traces are logged in the File Based Trace Log Publisher. See [Working with Logs and Log Publishers](#).

A Policy Decision Trace is an XML document that is formatted like the XACML policies. It demonstrates the sequence of steps taken by the policy engine to come to a decision for a specific request. The elements of the trace parallel the policies, policy targets, and policy rules that are evaluated. The following are included:

- The first line of the log entry identifies the message type as `POLICY-DECISION-TRACE`.
- The parameters of the XACML request being traced are listed, including the application, action, and resources.
- Following this is the trace itself, which is included in the `<DecisionTrace>` XACML element.

The trace also includes entries for each policy, rule, and target evaluated during the decision process. Each entry contains a `result` XML attribute, which specifies the result of evaluating the corresponding XACML element.

Troubleshooting Denied Access

Policies can issue [XACML AdviceExpressions](#) for any policy request that is denied. This passes additional information to the client as to the reason for denying access. Both the OAuth2 endpoints and the SCIM endpoint will look for error advice returned from the policy engine and include it in the error response generated for the client. If a policy denies a request without advice, the error response is `access_denied`.

The following error advice may be included in policy.

Policy Error Advice		
Advice ID	Attribute ID	Value
	error	Error identifier or code. For an OAuth2 response, this value populates the error parameter in the OAuth2 error response, as defined in the OAuth2 Authorization Framework. For SCIM responses, this value will be used to populate the <code>scimType error</code> parameter.
request-denied-reason	error_description	The value of the <code>error_description</code> parameter of an OAuth2 error response, or the detail parameter of a SCIM error response.

The following is an example of XACML Advice specifying that an `invalid_scope` error response should be returned:

```
<AdviceExpressions>
  <AdviceExpression AdviceId="request-denied-reason" AppliesTo="Deny">
    <AttributeAssignmentExpression AttributeId="error">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        invalid_scope
      </AttributeValue>
    </AttributeAssignmentExpression>
    <AttributeAssignmentExpression AttributeId="error-description">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        Application not authorized for requested scope
      </AttributeValue>
    </AttributeAssignmentExpression>
  </AdviceExpression>
</AdviceExpressions>
```

With this advice, the following error will be returned to the OAuth2 client:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer
error="invalid_scope",
error_description="Application not authorized for requested scope"
```

Unsupported XACML Features

When creating policies, the following XACML 3.0 features are *not* supported:

- **No support for embedded XML content in a request.** The following XACML elements related to XML processing have *not* been implemented:
 - `<PolicyDefaults>` and `<PolicySetDefaults>`
 - `<XPathVersion>`
 - XPath functions `xpath-node-count`, `xpath-node-equal`, and `xpath-node-match`.
- **No support for versioning of policies.** XACML incorporates the idea of maintaining multiple versions of a policy, such as policy "X" version 1.0 and policy "X" version 2.0. A policy set then can specify (by reference) which version of policy 'X' is to be applied

when evaluating a request. The Data Broker only allows for a single instance of policy X to be stored. It does not support referencing a particular version of that policy. The following XACML elements related to versioning are not supported:

- `<VersionMatchType>` in `PolicyIDReference` or `PolicySetIDReference` elements
- **Limited Support for Multi Requests.** XACML specifies several ways that a request for multiple decisions can be contained within a single request context, described in the XACML Multiple Decision Profile document. The Data Broker only supports one version of a multiple-decision request by using multiple `<Attributes>` of the same category in the request. In addition, Data Broker policies only support multiple instances of the Resources category. As a result the following XACML elements are not supported:
 - `<MultiRequests>`
 - `<RequestReference>`
 - `<AttributesReference>`
 - `CombinedDecision` attribute of the `<Request>` element
 - `xml:id` attribute of the `<Attributes>` element
- **No support for Attribute Issuer or Policy Issuer.** These features allow for the writing of policies that determine which other policies should be used when evaluating a request. For example, a request may be subject only to policies whose issuer (author) are from some trusted source. This is a second-order feature and not relevant for environments where all policies are equal as to their trustworthiness. The following XACML elements related to issuers are not supported:
 - `<PolicyIssuer>`
 - `Issuer` attribute of the `<Attribute>` element
- **No support for Policy and Rule Combiner Parameters.** A policy-combining algorithm is a rule for how the decisions rendered by multiple applicable policies are to be combined in order to form an ultimate decision by a policy set or the policy decision point as a whole. Similarly, a rule-combining algorithm is a rule for how the decisions rendered by multiple rules within a single policy are to be combined. The Policy and Rule Combiner Parameters are relevant only if custom rule-combining or policy-combining algorithms are in effect. Since the Data Broker does not currently support adding custom rule-combining or policy-combining algorithms, XACML elements for the associated Combiner Parameters are not supported:
 - `<CombinerParameters>`
 - `<RuleCombinerParameters>`

Chapter 8: Configuring XACML Policies

- `<PolicyCombinerParameters>`
- `<PolicySetCombinerParameters>`

Chapter 9: Advanced Configuration

The Data Broker's non-user data consists of data in the server configuration. Generally, data in the server configuration define an individual Data Broker instance, and can include its place in a server topology. Multiple server instances can be grouped in two ways to share or mirror configuration settings:

- Server Groups – Servers that are added to a server group in the global configuration can share configuration changes across the group, or not.
- Cluster – This is a topology management setting that enables a set of servers to be grouped by a functional purpose, and any change to one is mirrored to all. A master server verifies any configuration change before it is propagated to other servers in the group.

Note

All configuration objects and settings are described in the HTML Configuration Reference, which can be accessed from the Management Console or from the `<server-root>/docs/index.html` page. Information in this chapter highlights configuration of interest to a Data Broker installation. For complete configuration options and details, see the Configuration Reference.

Topics include:

[General Server Configuration](#)

[Data Broker Server Advanced Configuration](#)

[Configuring Data Broker Templates](#)

[Topology Management](#)

General Server Configuration

There are tools and settings that are common across all UnboundID servers. These enable monitoring and managing the server, configuring and sending alerts and alarms, and managing the server's communication with clients. These configuration objects can be changed at the local server, with the option to apply changes to servers in a group.

Available Configuration Tools

Available command-line configuration tools include:

Command-line Tools	
Tool	Description
backup	Run full or incremental backups on one or more Data Brokers. This utility also supports the use of a properties file to pass predefined command-line arguments.
base64	Encode raw data using the base64 algorithm or decode base64-encoded data back to its raw representation.
collect-support-data	Collect and package system information useful in troubleshooting problems. The information is packaged as a ZIP archive that can be sent to a technical support representative.
consent-admin	Manage a resource owner consent over the Data Broker REST API. Consent is authorized by a resource owner to allow access to resources by an application.
config-diff	Generate a summary of the configuration changes in a local or remote server instance. The tool can be used to compare configuration settings when troubleshooting issues, or when verifying configuration settings on new servers.
create-initial-broker-config	Create an initial Data Broker configuration.
create-rc-script	Create a Run Control (RC) script that can be used to start, stop, and restart the Data Broker on Unix-based systems.
dsconfig	View and edit the Data Broker configuration.
dsframework	Manage administrative server groups or the global administrative user accounts that are used to configure servers within server groups.
dsjavaproperties	Configure the JVM arguments used to run the Data Broker and its associated tools. Before launching the command, edit the properties file located in <code>config/java.properties</code> to specify the desired JVM arguments and the <code>JAVA_HOME</code> environment variable.
encryption-settings	Manage the server encryption settings database.
evaluate-policy	Request a policy decision from the Data Broker.
ldapmodify	Perform LDAP modify, add, delete, and modify DN operations in the Data Broker.
ldappasswordmodify	Perform LDAP password modify operations in the Data Broker.
ldapsearch	Perform LDAP search operations in the Data Broker.
ldif-diff	Compare the contents of two LDIF files, the output being an LDIF file needed to bring the source file in sync with the target.
ldifmodify	Apply a set of modify, add, and delete operations against data in an LDIF file.

Command-line Tools

Tool	Description
list-backends	List the backends and base DN's configured in the Data Broker.
manage-extension	Install or update extension bundles. An extension bundle is a package of extension (s) that utilize the Server SDK to extend the functionality of the Data Broker. Any added extensions require a server re-start.
oauth2-request	Performs OAuth2 requests on the Data Broker. This tool can be used to test OAuth2 functions of the Data Broker, and to manage OAuth2 tokens on behalf of registered applications.
prepare-external-store	Prepares the external data stores for the Data Broker. This is run as part of the <code>create-initial-broker-config</code> tool during installation. This tool creates the broker user account, sets the correct password, and configures the account with required privileges. It will also install the necessary schema required by the Data Broker.
remove-defunct-server	Removes a permanently unavailable Data Broker after it has been removed from its topology by the <code>uninstall</code> tool.
restore	Restore a backup of the Data Broker.
review-license	Review and/or accept the product license.
server-state	View information about the current state of the Data Broker processes.
start-broker	Start the Data Broker.
status	Display basic server information.
stop-broker	Stop or restart the Data Broker.
sum-file-sizes	Calculate the sum of the sizes for a set of files.

Using the dsconfig tool

The `dsconfig` tool, is used to view or edit the Data Broker configuration, and is parallel in functionality with the Management Console. This utility can be run in interactive mode, non-interactive mode, and batch mode. Interactive mode provides an intuitive, menu-driven interface for accessing and configuring the server.

To start `dsconfig` in interactive mode, enter the following command:

```
$ bin/dsconfig
```

The `dsconfig` tool provides a batching mechanism that reads multiple `dsconfig` invocations from a file and executes them sequentially. The batch file advantage is that it minimizes LDAP connections and JVM invocations required with scripting each call. To use batch mode to read and execute a series of commands in a batch file, enter the following command:

```
$ dsconfig --bindDN uid=admin,dc=company,dc=com \
  --bindPassword password \
  --no-prompt \
  --batch-file </path/to/config-batch.txt>
```

The `logs/config-audit.log` file can be used to review the configuration changes made to the Data Broker and use them in the batch file.

Administrative Accounts

Users that authenticate to the Config API or the Management Console are stored in `cn=Root` DNs, `cn=config`, not the user store that is configured for the Data Broker. These users must exist on all instances of the Data Broker to manage a Topology of servers. The `setup` tool automatically copies one administrative account when performing an installation from a peer, but if changed, the accounts must be synchronized.

Managing Root User Accounts

A default root user, `cn=Directory Manager`, is created during installation and is stored in the server's configuration file (under `cn=Root` DNs, `cn=config`). The root user is the LDAP-equivalent of a UNIX super-user account and inherits its read-write privileges from the default root privilege set. Root user entries are stored in the server's configuration and not in backend data.

To limit full access, create separate administrator user accounts with limited privileges. Having separate user accounts for each administrator also makes it possible to enable password policy functionality (such as password expiration, password history, and requiring secure authentication) for each administrator.

Default Root Privileges

The UnboundID servers have a privilege subsystem that allows for a more fine-grained control of privilege assignments. The following set of root privileges are available to each root user DN:

Default Root Privileges	
Privilege	Description
<code>audit-data-security</code>	Allows the associated user to execute data security auditing tasks.
<code>backend-backup</code>	Allows the user to perform backend backup operations.
<code>backend-restore</code>	Allows the user to perform backend restore operations.
<code>bypass-acl</code>	Allows the user to bypass access control evaluation.
<code>config-read</code>	Allows the user to read the server configuration.
<code>config-write</code>	Allows the user to update the server configuration.
<code>disconnect-client</code>	Allows the user to terminate arbitrary client connections.
<code>ldif-export</code>	Allows the user to perform LDIF export operations.
<code>ldif-import</code>	Allows the user to perform LDIF import operations.
<code>lockdown-mode</code>	Allows the user to request a server lockdown.
<code>manage-topology</code>	Allows the user to make configuration changes to the topology.
<code>modify-acl</code>	Allows the user to modify access control rules.
<code>password-reset</code>	Allows the user to reset user passwords but not their own. The user must also have privileges granted by access control to write the user password to the target entry.

Default Root Privileges

Privilege	Description
permit-get-password-policy-state-issues	Allows the user to retrieve the state of a user account.
privilege-change	Allows the user to change the set of privileges for a specific user, or to change the set of privileges automatically assigned to a root user.
server-restart	Allows the user to request a server restart.
server-shutdown	Allows the user to request a server shutdown.
soft-delete-read	Allows the user access to soft-deleted entries.
stream-values	Allows the user to perform a stream values extended operation that obtains all entry DNs and/or all values for one or more attributes for a specified portion of the DIT.
third-party-task	Allows the associated user to invoke tasks created by third-party developers.
unindexed-search	Allows the user to perform an unindexed search in the Oracle Berkeley DB Java Edition backend.
update-schema	Allows the user to update the server schema.
use-admin-session	Allows the associated user to use an administrative session to request that operations be processed using a dedicated pool of worker threads.

Creating a Root User

Additional root users can be added under `cn=Root DNs,cn=config`. Whether the new root DN inherits the default set of root privileges is determined by the value of `ds-cfginherit-default-root-privileges`.

The following steps need to be repeated on every server in the topology.

1. Open a text editor, and create a file containing the root user entry.

```
dn: cn=Data Admin,cn=Root DNs,cn=config
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: uidPerson
objectClass: ds-cfg-root-dn-user
userPassword: password
cn: Data Admin
sn: Admin
ds-cfg-alternate-bind-dn: cn=Data Admin
givenName: Data
ds-cfg-inherit-default-root-privileges: false
ds-privilege-name: bypass-acl
ds-privilege-name: password-reset
ds-privilege-name: update-schema
ds-privilege-name: unindexed-search
ds-privilege-name: use-admin-session
ds-privilege-name: manage-topology
```

2. Use `ldapmodify` to add the entry.

```
$ bin/ldapmodify --port <ldap-port> \  
  --bindDN "cn=Directory Manager" \  
  --bindPassword <password> \  
  --defaultAdd \  
  --filename "rootuser.ldif"
```

Modifying the Root User Password

To change a root user's password, use the `ldappasswordmodify` tool.

1. Open a text editor and create a text file containing the new password, such as `rootuser.txt`.

```
$ echo password > rootuser.txt
```

2. Use `ldappasswordmodify` to change the root user's password.

```
$ bin/ldappasswordmodify \  
  --port <ldap-port> \  
  --bindDN "cn=Directory Manager" \  
  --bindPassword <password> \  
  --newPasswordFile rootuser.txt
```

3. Remove the text file.

Using the Configuration API

UnboundID servers provide a Configuration API, which may be useful in situations where using LDAP to update the server configuration is not possible. The API is consistent with the System for Cross-domain Identity Management (SCIM) 2.0 protocol and uses JSON as a text exchange format, so all request headers should allow the `application/json` content type.

The server includes a servlet extension that provides read and write access to the server's configuration over HTTP. The extension is enabled by default for new installations, and can be enabled for existing deployments by simply adding the extension to one of the server's HTTP Connection Handlers, as follows:

```
$ bin/dsconfig set-connection-handler-prop \  
  --handler-name "HTTPS Connection Handler" \  
  --add http-servlet-extension:Configuration
```

The API is made available on the HTTPS Connection handler's `host:port` in the `/config` context. Due to the potentially sensitive nature of the server's configuration, the HTTPS Connection Handler should be used, for hosting the Configuration extension.

Authentication and Authorization

Clients must use HTTP Basic authentication to authenticate to the Configuration API. If the username value is not a DN, then it will be resolved to a DN value using the identity mapper associated with the Configuration servlet. By default, the Configuration API uses an identity mapper that allows an entry's UID value to be used as a username. To customize this behavior, either customize the default identity mapper, or specify a different identity mapper using the Configuration servlet's `identity-mapper` property. For example:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name Configuration \
  --set "identity-mapper:Alternative Identity Mapper"
```

To access configuration information, users must have the appropriate privileges:

- To access the `cn=config` backend, users must have the `bypass-acl` privilege or be allowed access to the configuration using an ACI.
- To read configuration information, users must have the `config-read` privilege.
- To update the configuration, users must have the `config-write` privilege.

Relationship Between the Configuration API and the dsconfig Tool

The Configuration API is designed to mirror the `dsconfig` tool, using the same names for properties and object types. Property names are presented as hyphen case in `dsconfig` and as camel-case attributes in the API. In API requests that specify property names, case is not important. Therefore, `baseDN` is the same as `baseDn`. Object types are represented in hyphen case. API paths mirror what is in `dsconfig`. For example, the `dsconfig list-connection-handlers` command is analogous to the API's `/config/connection-handlers` path. Object types that appear in the schema URNs adhere to a `type:subtype` syntax. For example, a Local DB Backend's schema URN is `urn:unboundid:schemas:configuration:2.0:backend:local-db`. Like the `dsconfig` tool, all configuration updates made through the API are recorded in `logs/config-audit.log`.

The API includes the filter, sort, and pagination query parameters described by the SCIM specification. Specific attributes may be requested using the `attributes` query parameter, whose value must be a comma-delimited list of properties to be returned, for example `attributes=baseDN,description`. Likewise, attributes may be excluded from responses by specifying the `excludedAttributes` parameter. See [Sorting and Filtering with the Configuration API](#) for more information on query parameters.

Operations supported by the API are those typically found in REST APIs:

HTTP Method	Description	Related dsconfig Example
GET	Lists the attributes of an object when used with a path representing an object, such as <code>/config/global-configuration</code> or <code>/config/backends/userRoot</code> . Can also list objects when used with a path representing a parent relation, such as <code>/config/backends</code> .	get-backend-prop list-backends get-global-configuration-prop
POST	Creates a new instance of an object when used with a relation parent path, such as <code>config/backends</code> .	create-backend
PUT	Replaces the existing attributes of an object. A PUT operation is similar to a PATCH operation, except that the PATCH is determined by determining the difference between an existing target object and a supplied source object. Only those attributes in the source object are modified in the target object. The target object is specified using a path, such as <code>/config/backends/userRoot</code> .	set-backend-prop set-global-configuration-prop
PATCH	Updates the attributes of an existing object when used with a path representing an object, such as <code>/config/backends/userRoot</code> . See PATCH Example .	set-backend-prop set-global-configuration-prop
DELETE	Deletes an existing object when used with a path representing an object, such as <code>/config/backends/userRoot</code> .	delete-backend

The OPTIONS method can also be used to determine the operations permitted for a particular path.

Object names, such as `userRoot` in the Description column, must be URL-encoded in the path segment of a URL. For example, `%20` must be used in place of spaces, and `%25` is used in place of the percent (%) character. So the URL for accessing the HTTP Connection Handler object is:

```
/config/connection-handlers/http%20connection%20handler
```

GET Example

The following is a sample GET request for information about the `userRoot` backend:

```
GET /config/backends/userRoot
Host: example.com:5033
Accept: application/scim+json
```

The response:

```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
  ],
  "id": "userRoot",
  "meta": {
    "resourceType": "Local DB Backend",
    "location": "http://localhost:5033/config/backends/userRoot"
  },
  "backendID": "userRoot2",
  "backgroundPrime": "false",
```

```

"backupFilePermissions": "700",
"baseDN": [
  "dc=example2,dc=com"
],
"checkpointOnCloseCount": "2",
"cleanerThreadWaitTime": "120000",
"compressEntries": "false",
"continuePrimeAfterCacheFull": "false",
"dbBackgroundSyncInterval": "1 s",
"dbCachePercent": "10",
"dbCacheSize": "0 b",
"dbCheckpointInterval": "20 mb",
"dbCheckpointHighPriority": "false",
"dbCheckpointWakeupInterval": "1 m",
"dbCleanOnExplicitGC": "false",
"dbCleanerMinUtilization": "75",
"dbCompactKeyPrefixes": "true",
"dbDirectory": "db",
"dbDirectoryPermissions": "700",
"dbEvictorCriticalPercentage": "0",
"dbEvictorLruOnly": "false",
"dbEvictorNodesPerScan": "10",
"dbFileCacheSize": "1000",
"dbImportCachePercent": "60",
"dbLogFileMax": "50 mb",
"dbLoggingFileHandlerOn": "true",
"dbLoggingLevel": "CONFIG",
"dbNumCleanerThreads": "0",
"dbNumLockTables": "0",
"dbRunCleaner": "true",
"dbTxnNoSync": "false",
"dbTxnWriteNoSync": "true",
"dbUseThreadLocalHandles": "true",
"deadlockRetryLimit": "10",
"defaultCacheMode": "cache-keys-and-values",
"defaultTxnMaxLockTimeout": "10 s",
"defaultTxnMinLockTimeout": "10 s",
"enabled": "false",
"explodedIndexEntryThreshold": "4000",
"exportThreadCount": "0",
"externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
"externalTxnDefaultMaxLockTimeout": "100 ms",
"externalTxnDefaultMinLockTimeout": "100 ms",
"externalTxnDefaultRetryAttempts": "2",
"hashEntries": "false",
"id2childrenIndexEntryLimit": "66",
"importTempDirectory": "import-tmp",
"importThreadCount": "16",
"indexEntryLimit": "4000",
"isPrivateBackend": "false",
"javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
"jeProperty": [
  "je.cleaner.adjustUtilization=false",
  "je.nodeMaxEntries=32"
],
"numRecentChanges": "50000",
"offlineProcessDatabaseOpenTimeout": "1 h",

```

Chapter 9: Advanced Configuration

```
"primeAllIndexes": "true",
"primeMethod": [
  "none"
],
"primeThreadCount": "2",
"primeTimeLimit": "0 ms",
"processFiltersWithUndefinedAttributeTypes": "false",
"returnUnavailableForUntrustedIndex": "true",
"returnUnavailableWhenDisabled": "true",
"setDegradedAlertForUntrustedIndex": "true",
"setDegradedAlertWhenDisabled": "true",
"subtreeDeleteBatchSize": "5000",
"subtreeDeleteSizeLimit": "5000",
"uncachedId2entryCacheMode": "cache-keys-only",
"writabilityMode": "enabled"
}
```

GET List Example

The following is a sample GET request for all local backends:

```
GET /config/backends
Host: example.com:5033
Accept: application/scim+json
```

The response (which has been shortened):

```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:ListResponse"
  ],
  "totalResults": 24,
  "Resources": [
    {
      "schemas": [
        "urn:unboundid:schemas:configuration:2.0:backend:ldif"
      ],
      "id": "adminRoot",
      "meta": {
        "resourceType": "LDIF Backend",
        "location": "http://localhost:5033/config/backends/adminRoot"
      },
      "backendID": "adminRoot",
      "backupFilePermissions": "700",
      "baseDN": [
        "cn=admin data"
      ],
      "enabled": "true",
      "isPrivateBackend": "true",
      "javaClass": "com.unboundid.directory.server.backends.LDIFBackend",
      "ldifFile": "config/admin-backend.ldif",
      "returnUnavailableWhenDisabled": "true",
      "setDegradedAlertWhenDisabled": "false",
      "writabilityMode": "enabled"
    },
    {
      "schemas": [
        "urn:unboundid:schemas:configuration:2.0:backend:trust-store"
      ]
    }
  ]
}
```



```

    ],
    "id": "ads-truststore",
    "meta": {
      "resourceType": "Trust Store Backend",
      "location": "http://localhost:5033/config/backends/ads-truststore"
    },
    "backendID": "ads-truststore",
    "backupFilePermissions": "700",
    "baseDN": [
      "cn=ads-truststore"
    ],
    "enabled": "true",
    "javaClass": "com.unboundid.directory.server.backends.TrustStoreBackend",
    "returnUnavailableWhenDisabled": "true",
    "setDegradedAlertWhenDisabled": "true",
    "trustStoreFile": "config/server.keystore",
    "trustStorePin": "*****",
    "trustStoreType": "JKS",
    "writabilityMode": "enabled"
  },
  {
    "schemas": [
      "urn:unboundid:schemas:configuration:2.0:backend:alarm"
    ],
    "id": "alarms",
    "meta": {
      "resourceType": "Alarm Backend",
      "location": "http://localhost:5033/config/backends/alarms"
    },
  },
  ...

```

PATCH Example

Configuration can be modified using the HTTP PATCH method. The PATCH request body is a JSON object formatted according to the SCIM patch request. The Configuration API, supports a subset of possible values for the `path` attribute, used to indicate the configuration attribute to modify.

The configuration object's attributes can be modified in the following ways. These operations are analogous to the `dsconfig modify-[object]` options.

- An operation to set the single-valued `description` attribute to a new value:

```

{
  "op" : "replace",
  "path" : "description",
  "value" : "A new backend."
}

```

is analogous to:

```

$ dsconfig set-backend-prop --backend-name userRoot \
  --set "description:A new backend"

```

- An operation to add a new value to the multi-valued `jeProperty` attribute:

Chapter 9: Advanced Configuration

```
{
  "op" : "add",
  "path" : "jeProperty",
  "value" : "je.env.backgroundReadLimit=0"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --add je-property:je.env.backgroundReadLimit=0
```

- An operation to remove a value from a multi-valued property. In this case, `path` specifies a SCIM filter identifying the value to remove:

```
{
  "op" : "remove",
  "path" : "[jeProperty eq \"je.cleaner.adjustUtilization=false\"]"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --remove je-property:je.cleaner.adjustUtilization=false
```

- A second operation to remove a value from a multi-valued property, where the `path` specifies both an attribute to modify, and a SCIM filter whose attribute is `value`:

```
{
  "op" : "remove",
  "path" : "jeProperty[value eq \"je.nodeMaxEntries=32\"]"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --remove je-property:je.nodeMaxEntries=32
```

- An option to remove one or more values of a multi-valued attribute. This has the effect of restoring the attribute's value to its default value:

```
{
  "op" : "remove",
  "path" : "id2childrenIndexEntryLimit"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --reset id2childrenIndexEntryLimit
```

The following is the full example request. The API responds with the entire modified configuration object, which may include a SCIM extension attribute `urn:unboundid:schemas:configuration:messages` containing additional instructions:

Example request:

```
PATCH /config/backends/userRoot
Host: example.com:5033
```

Accept: application/scim+json

```
{
  "schemas" : [ "urn:ietf:params:scim:api:messages:2.0:PatchOp" ],
  "Operations" : [ {
    "op" : "replace",
    "path" : "description",
    "value" : "A new backend."
  }, {
    "op" : "add",
    "path" : "jeProperty",
    "value" : "je.env.backgroundReadLimit=0"
  }, {
    "op" : "remove",
    "path" : "[jeProperty eq \"je.cleaner.adjustUtilization=false\"]"
  }, {
    "op" : "remove",
    "path" : "jeProperty[value eq \"je.nodeMaxEntries=32\"]"
  }, {
    "op" : "remove",
    "path" : "id2childrenIndexEntryLimit"
  } ]
}
```

Example response:

```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
  ],
  "id": "userRoot2",
  "meta": {
    "resourceType": "Local DB Backend",
    "location": "http://example.com:5033/config/backends/userRoot2"
  },
  "backendID": "userRoot2",
  "backgroundPrime": "false",
  "backupFilePermissions": "700",
  "baseDN": [
    "dc=example2,dc=com"
  ],
  "checkpointOnCloseCount": "2",
  "cleanerThreadWaitTime": "120000",
  "compressEntries": "false",
  "continuePrimeAfterCacheFull": "false",
  "dbBackgroundSyncInterval": "1 s",
  "dbCachePercent": "10",
  "dbCacheSize": "0 b",
  "dbCheckpointIntervalBytes": "20 mb",
  "dbCheckpointHighPriority": "false",
  "dbCheckpointWakeupInterval": "1 m",
  "dbCleanOnExplicitGC": "false",
  "dbCleanerMinUtilization": "75",
  "dbCompactKeyPrefixes": "true",
  "dbDirectory": "db",
  "dbDirectoryPermissions": "700",
  "dbEvictorCriticalPercentage": "0",
  "dbEvictorLruOnly": "false",

```

Chapter 9: Advanced Configuration

```
"dbEvictorNodesPerScan": "10",
"dbFileCacheSize": "1000",
"dbImportCachePercent": "60",
"dbLogFileMax": "50 mb",
"dbLoggingFileHandlerOn": "true",
"dbLoggingLevel": "CONFIG",
"dbNumCleanerThreads": "0",
"dbNumLockTables": "0",
"dbRunCleaner": "true",
"dbTxnNoSync": "false",
"dbTxnWriteNoSync": "true",
"dbUseThreadLocalHandles": "true",
"deadlockRetryLimit": "10",
"defaultCacheMode": "cache-keys-and-values",
"defaultTxnMaxLockTimeout": "10 s",
"defaultTxnMinLockTimeout": "10 s",
"description": "123",
"enabled": "false",
"explodedIndexEntryThreshold": "4000",
"exportThreadCount": "0",
"externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
"externalTxnDefaultMaxLockTimeout": "100 ms",
"externalTxnDefaultMinLockTimeout": "100 ms",
"externalTxnDefaultRetryAttempts": "2",
"hashEntries": "false",
"importTempDirectory": "import-tmp",
"importThreadCount": "16",
"indexEntryLimit": "4000",
"isPrivateBackend": "false",
"javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
"jeProperty": [
  "\"je.env.backgroundReadLimit=0\"",
],
"numRecentChanges": "50000",
"offlineProcessDatabaseOpenTimeout": "1 h",
"primeAllIndexes": "true",
"primeMethod": [
  "none"
],
"primeThreadCount": "2",
"primeTimeLimit": "0 ms",
"processFiltersWithUndefinedAttributeTypes": "false",
"returnUnavailableForUntrustedIndex": "true",
"returnUnavailableWhenDisabled": "true",
"setDegradedAlertForUntrustedIndex": "true",
"setDegradedAlertWhenDisabled": "true",
"subtreeDeleteBatchSize": "5000",
"subtreeDeleteSizeLimit": "5000",
"uncachedId2entryCacheMode": "cache-keys-only",
"writabilityMode": "enabled",
"urn:unboundid:schemas:configuration:messages:2.0": {
  "requiredActions": [
    {
      "property": "jeProperty",
      "type": "componentRestart",
      "synopsis": "In order for this modification to take effect,
```

```

        the component must be restarted, either by disabling and
        re-enabling it, or by restarting the server"
    },
    {
        "property": "id2childrenIndexEntryLimit",
        "type": "other",
        "synopsis": "If this limit is increased, then the contents
        of the backend must be exported to LDIF and re-imported to
        allow the new limit to be used for any id2children keys
        that had already hit the previous limit."
    }
]
}
}
}

```

API Paths

The Configuration API is available under the `/config` path. A full listing of root sub-paths can be obtained from the `/config/ResourceTypes` endpoint:

```

GET /config/ResourceTypes
Host: example.com:5033
Accept: application/scim+json

```

Sample response (abbreviated):

```

{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:ListResponse"
  ],
  "totalResults": 520,
  "Resources": [
    {
      "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:ResourceType"
      ],
      "id": "dsee-compat-access-control-handler",
      "name": "DSEE Compat Access Control Handler",
      "description": "The DSEE Compat Access Control
        Handler provides an implementation that uses syntax
        compatible with the Sun Java System Directory Server
        Enterprise Edition access control handler.",
      "endpoint": "/access-control-handler",
      "meta": {
        "resourceType": "ResourceType",
        "location": "http://example.com:5033/config/ResourceTypes/dsee-compat-access-
control-handler"
      }
    },
    {
      "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:ResourceType"
      ],
      "id": "access-control-handler",
      "name": "Access Control Handler",
      "description": "Access Control Handlers manage the

```

Chapter 9: Advanced Configuration

```
application-wide access control. The server's access
control handler is defined through an extensible
interface, so that alternate implementations can be created.
Only one access control handler may be active in the server
at any given time.",
"endpoint": "/access-control-handler",
"meta": {
  "resourceType": "ResourceType",
  "location": "http://example.com:5033/config/ResourceTypes/access-control-handler"
},
{
  ...
```

The response's `endpoint` elements enumerate all available sub-paths. The path `/config/access-control-handler` in the example can be used to get a list of existing access control handlers, and create new ones. A path containing an object name like `/config/backends/{backendName}`, where `{backendName}` corresponds to an existing backend (such as `userRoot`) can be used to obtain an object's properties, update the properties, or delete the object.

Some paths reflect hierarchical relationships between objects. For example, properties of a local DB VLV index for the `userRoot` backend are available using a path like `/config/backends/userRoot/local-db-indexes/uid`. Some paths represent singleton objects, which have properties but cannot be deleted nor created. These paths can be differentiated from others by their singular, rather than plural, relation name (for example `global-configuration`).

Sorting and Filtering Configuration Objects

The Configuration API supports SCIM parameters for filter, sorting, and pagination. Search operations can specify a SCIM filter used to narrow the number of elements returned. See the SCIM specification for the full set of operations for SCIM filters. Clients may also specify sort parameters, or paging parameters. As previously mentioned, clients may specify attributes to include or exclude in both get and list operations.

GET Parameters for Sorting and Filtering	
GET Parameter	Description
filter	Values can be simple SCIM filters such as <code>id eq "userRoot"</code> or compound filters like <code>meta.resourceType eq "Local DB Backend"</code> and <code>baseDn co "dc=example,dc=com"</code> .
sortBy	Specifies a property value by which to sort.
sortOrder	Specifies either <code>ascending</code> or <code>descending</code> alphabetical order.
startIndex	1-based index of the first result to return.
count	Indicates the number of results per page.

Updating Properties

The Configuration API supports the HTTP PUT method as an alternative to modifying objects with HTTP PATCH. With PUT, the server computes the differences between the object in the request with the current version in the server, and performs modifications where necessary. The server will never remove attributes that are not specified in the request. The API responds with the entire modified object.

Request:

```
PUT /config/backends/userRoot
Host: example.com:5033
Accept: application/scim+json
{
  "description" : "A new description."
}
```

Response:

```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
  ],
  "id": "userRoot",
  "meta": {
    "resourceType": "Local DB Backend",
    "location": "http://example.com:5033/config/backends/userRoot"
  },
  "backendID": "userRoot",
  "backgroundPrime": "false",
  "backupFilePermissions": "700",
  "baseDN": [
    "dc=example,dc=com"
  ],
  "checkpointOnCloseCount": "2",
  "cleanerThreadWaitTime": "120000",
  "compressEntries": "false",
  "continuePrimeAfterCacheFull": "false",
  "dbBackgroundSyncInterval": "1 s",
  "dbCachePercent": "25",
  "dbCacheSize": "0 b",
  "dbCheckpointIntervalBytes": "20 mb",
  "dbCheckpointIntervalHighPriority": "false",
  "dbCheckpointIntervalWakeup": "30 s",
  "dbCleanOnExplicitGC": "false",
  "dbCleanerMinUtilization": "75",
  "dbCompactKeyPrefixes": "true",
  "dbDirectory": "db",
  "dbDirectoryPermissions": "700",
  "dbEvictorCriticalPercentage": "5",
  "dbEvictorLruOnly": "false",
  "dbEvictorNodesPerScan": "10",
  "dbFileCacheSize": "1000",
  "dbImportCachePercent": "60",
  "dbLogFileMax": "50 mb",
  "dbLoggingFileHandlerOn": "true",
  "dbLoggingLevel": "CONFIG",
```

Chapter 9: Advanced Configuration

```
"dbNumCleanerThreads": "1",
"dbNumLockTables": "0",
"dbRunCleaner": "true",
"dbTxnNoSync": "false",
"dbTxnWriteNoSync": "true",
"dbUseThreadLocalHandles": "true",
"deadlockRetryLimit": "10",
"defaultCacheMode": "cache-keys-and-values",
"defaultTxnMaxLockTimeout": "10 s",
"defaultTxnMinLockTimeout": "10 s",
"description": "abc",
"enabled": "true",
"explodedIndexEntryThreshold": "4000",
"exportThreadCount": "0",
"externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
"externalTxnDefaultMaxLockTimeout": "100 ms",
"externalTxnDefaultMinLockTimeout": "100 ms",
"externalTxnDefaultRetryAttempts": "2",
"hashEntries": "true",
"importTempDirectory": "import-tmp",
"importThreadCount": "16",
"indexEntryLimit": "4000",
"isPrivateBackend": "false",
"javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
"numRecentChanges": "50000",
"offlineProcessDatabaseOpenTimeout": "1 h",
"primeAllIndexes": "true",
"primeMethod": [
  "none"
],
"primeThreadCount": "2",
"primeTimeLimit": "0 ms",
"processFiltersWithUndefinedAttributeTypes": "false",
"returnUnavailableForUntrustedIndex": "true",
"returnUnavailableWhenDisabled": "true",
"setDegradedAlertForUntrustedIndex": "true",
"setDegradedAlertWhenDisabled": "true",
"subtreeDeleteBatchSize": "5000",
"subtreeDeleteSizeLimit": "100000",
"uncachedId2entryCacheMode": "cache-keys-only",
"writabilityMode": "enabled"
}
```

Administrative Actions

Updating a property may require an administrative action before the change can take effect. If so, the server will return 200 Success, and any actions are returned in the `urn:unboundid:schemas:configuration:messages:2.0` section of the JSON response that represents the entire object that was created or modified.

For example, changing the `jeProperty` of a backend will result in the following:

```
"urn:unboundid:schemas:configuration:messages:2.0": {
  "requiredActions": [
    {
      "property": "baseContextPath",
```



```

    "type": "componentRestart",
    "synopsis": "In order for this modification to
        take effect, the component must be restarted,
        either by disabling and re-enabling it, or by
        restarting the server"
  },
  {
    "property": "id2childrenIndexEntryLimit",
    "type": "other",
    "synopsis": "If this limit is increased, then the
        contents of the backend must be exported to LDIF
        and re-imported to allow the new limit to be used
        for any id2children keys that had already hit the
        previous limit."
  }
]
}
...

```

Updating Servers and Server Groups

Servers can be configured as part of a server group, so that configuration changes that are applied to a single server, are then applied to all servers in a group. When managing a server that is a member of a server group, creating or updating objects using the Configuration API requires the `applyChangeTo` query parameter. The behavior and acceptable values for this parameter are identical to the `dsconfig` parameter of the same name. A value of `singleServer` or `serverGroup` can be specified. For example:

```
https://example.com:5033/config/Backends/userRoot?applyChangeTo=singleServer
```

Note

This does not apply to mirrored subtree objects, which include Topology and Cluster level objects. Changes made to mirrored objects are applied to all objects in the subtree.

Configuration API Responses

Clients of the API should examine the HTTP response code in order to determine the success or failure of a request. The following are response codes and their meanings:

Response Code	Description	Response Body
200 Success	The requested operation succeeded, with the response body being the configuration object that was created or modified. If further actions are required, they are included in the <code>urn:unboundid:schemas:configuration:messages:2.0</code> object.	List of objects, or object properties, administrative actions.
204 No Content	The requested operation succeeded and no further information has been provided, such as in the case of a DELETE operation.	None.
400 Bad Request	The request contents are incorrectly formatted or a request is made for an invalid API version.	Error summary and optional message.

Response Code	Description	Response Body
401 Unauthorized	User authentication is required. Some user agents such as browsers may respond by prompting for credentials. If the request had specified credentials in an Authorization header, they are invalid.	None.
403 Forbidden	The requested operation is forbidden either because the user does not have sufficient privileges or some other constraint such as an object is edit-only and cannot be deleted.	None.
404 Not Found	The requested path does not refer to an existing object or object relation.	Error summary and optional message.
409 Conflict	The requested operation could not be performed due to the current state of the configuration. For example, an attempt was made to create an object that already exists or an attempt was made to delete an object that is referred to by another object.	Error summary and optional message.
415 Unsupported Media Type	The request is such that the Accept header does not indicate that JSON is an acceptable format for a response.	None.
500 Server Error	The server encountered an unexpected error. Please report server errors to customer support.	Error summary and optional message.

An application that uses the Configuration API should limit dependencies on particular text appearing in error message content. These messages may change, and their presence may depend on server configuration. Use the HTTP return code and the context of the request to create a client error message. The following is an example encoded error message:

```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:Error"
  ],
  "status": 404,
  "scimType": null,
  "detail": "The Local DB Index does not exist."
}
```

Domain Name Service (DNS) Caching

If needed, two global configuration properties can be used to control the caching of hostname-to-numeric IP address (DNS lookup) results returned from the name resolution services of the underlying operating system. Use the `dsconfig` tool to configure these properties.

network-address-cache-ttl – Sets the Java system property `networkaddress.cache.ttl`, and controls the length of time in seconds that a hostname-to-IP address mapping can be cached. The default behavior is to keep resolution results for one hour (3600 seconds). This setting applies to the server and all extensions loaded by the server.

network-address-outage-cache-enabled – Caches hostname-to-IP address results in the event of a DNS outage. This is set to `true` by default, meaning name resolution results are cached. Unexpected service interruptions may occur during planned or unplanned maintenance, network outages or an infrastructure attack. This cache may allow the server to

function during a DNS outage with minimal impact. This cache is not available to server extensions.

IP Address Reverse Name Lookups

UnboundID servers do not explicitly perform numeric IP address-to-hostname lookups. However address masks configured in Access Control Lists (ACIs), Connection Handlers, Connection Criteria, and Certificate handshake processing may trigger implicit reverse name lookups. For more information about how address masks are configured in the server, review the following information for each server:

- ACI dns: bind rules under *Managing Access Control* (Data Store and Proxy Servers)
- ds-auth-allowed-address: *Adding Operational Attributes that Restrict Authentication* (Data Store)
- Connection Criteria: *Restricting Server Access Based on Client IP Address* (Data Store and Proxy Servers)
- Connection Handlers: restrict server access using Connection Handlers (Configuration Reference Guide for all servers)

System Alarms, Alerts, and Gauges

UnboundID servers provide tools to monitor and manage the health of the system. The Data Broker provides delivery mechanisms (handlers) for administrative alerts using JMX or SNMP, in addition to standard error logging. All can be configured with the `dsconfig` tool.

Alerts and alarms reflect state changes within the server that may be of interest to a user or monitoring service. An alarm represents a stateful condition of the server or a resource that may indicate a problem, such as low disk space or external server unavailability. A gauge defines a set of threshold values with a specified severity that, when crossed, cause the server to enter or exit an alarm state. Gauges are used for monitoring continuous values like CPU load or free disk space (Numeric Gauge), or an enumerated set of values such as 'server available' or 'server unavailable' (Indicator Gauge). Gauges generate alarms, when the gauge's severity changes due to changes in the monitored value. Like alerts, alarms have severity (NORMAL, WARNING, MINOR, MAJOR, CRITICAL), name, and message. Alarms will always have a Condition property, and may have a Specific Problem or Resource property. If surfaced through SNMP, a Probable Cause property and Alarm Type property are also listed. Alarms can be configured to generate alerts when the alarm's severity changes.

There are two alert types supported by the server - standard and alarm-specific. The server constantly monitors for conditions that may attention by administrators, such as low disk space. For this condition, the standard alert is `low-disk-space-warning`, and the alarm-specific alert is `alarm-warning`. The server can be configured to generate alarm-specific alerts instead of, or in addition to, standard alerts. By default, standard alerts are generated for conditions internally monitored by the server. However, gauges can only generate alarm-alerts.

The server installs gauges for CPU, disk, and memory usage that can be cloned or configured through the `dsconfig` tool. Existing gauges can be tailored to fit each environment by adjusting the update interval and threshold values. Configuration of system gauges determines the criteria by which alarms are triggered. The Stats Logger can be used to view historical information about the value and severity of all system gauges.

The server is compliant with the International Telecommunication Union CCITT Recommendation X.733 (1992) standard for generating and clearing alarms. If configured, entering or exiting an alarm state can result in one or more alerts. An alarm state is exited when the condition no longer applies. An `alarm_cleared` alert type is generated by the system when an alarm's severity changes from a non-normal severity to any other severity. An `alarm_cleared` alert will correlate to a previous alarm when Condition and Resource property are the same. The Alarm Manager, which governs the actions performed when an alarm state is entered, is configurable through the `dsconfig` tool.

Like the Alerts Backend, which stores information in `cn=alerts`, the Alarm Backend stores information within the `cn=alarms` backend. Unlike alerts, alarm thresholds have a state over time that can change in severity and be cleared when a monitored value returns to normal. Alarms can be viewed with the `status` tool.

As with other alert types, alert handlers can be configured to manage the alerts generated by alarms. A complete listing of system alerts, alarms, and their severity is available in `<server-root>/docs/admin-alerts-list.csv`.

Alert Handlers

Alert notifications can be sent to administrators when significant problems or events occur during processing, such as problems during server startup or shutdown. The Data Broker provides a number of alert handler implementations configured with the `dsconfig` tool or the Management Console, including:

- **Error Log Alert Handler** – Sends administrative alerts to the configured server error logger(s).
- **JMX Alert Handler** – Sends administrative alerts to clients using the Java Management Extensions (JMX) protocol. The server uses JMX for monitoring entries and requires that the JMX connection handler be enabled.
- **SNMP Alert Handler** – Sends administrative alerts to clients using the Simple Network Monitoring Protocol (SNMP). The server must have an SNMP agent capable of communicating via SNMP 2c.

If needed, the Server SDK can be used to implement additional, third-party alert handlers.

Test Alarms and Alerts

After gauges, alarms, and alert handlers are configured, verify that the server takes the appropriate action when an alarm state changes by manually increasing the severity of a gauge. Alarms and alerts can be verified with the `status` tool.

Perform the following steps to test alarms and alerts:

1. Configure a gauge with `dsconfig` and set the `override-severity` property to `critical`. The following example uses the CPU Usage (Percent) gauge.

```
$ dsconfig set-gauge-prop \
  --gauge-name "CPU Usage (Percent)" \
  --set override-severity:critical
```

2. Run the `status` tool to verify that an alarm was generated with corresponding alerts. The `status` tool provides a summary of the server's current state with key metrics and a list of recent alerts and alarms. The sample output has been shortened to show just the alarms and alerts information.

```
$ bin/status
```

```

          --- Administrative Alerts ---
Severity : Time      : Message
-----:-----:-----
Error    : 11/Aug/2015 : Alarm [CPU Usage (Percent). Gauge CPU Usage (Percent)
          : 15:41:00      : for Host System Recent CPU and Memory has
          : -0500           : a current value of '18.58333333333332'.
          :                 : The severity is currently OVERRIDDEN in the
          :                 : Gauge's configuration to 'CRITICAL'.
          :                 : The actual severity is: The severity is
          :                 : currently 'NORMAL', having assumed this severity
          :                 : Mon Aug 11 15:41:00 CDT 2015. If CPU use is high,
          :                 : check the server's current workload and make any
          :                 : needed adjustments. Reducing the load on the system
          :                 : will lead to better response times.
          :                 : Resource='Host System']
          :                 : raised with critical severity
Shown are alerts of severity [Info,Warning,Error,Fatal] from the past 48 hours
Use the --maxAlerts and/or --alertSeverity options to filter this list

```

```

          --- Alarms ---
Severity : Severity   : Condition : Resource   : Details
          : Start Time :           :           :
-----:-----:-----:-----:-----
Critical : 11/Aug/2015: CPU Usage : Host System : Gauge CPU Usage (Percent) for
          : 15:41:00    : (Percent) : Host System
          : -0500       :           : has a current value of
          :             :           : '18.785714285714285'.
          :             :           : The severity is currently
          :             :           : 'CRITICAL', having assumed
          :             :           : this severity Mon Aug 11
          :             :           : 15:49:00 CDT 2015. If CPU use
          :             :           : is high, check the server's
          :             :           : current workload and make any
          :             :           : needed adjustments. Reducing
          :             :           : the load on the system will
          :             :           : lead to better response times
Shown are alarms of severity [Warning,Minor,Major,Critical]
Use the --alarmSeverity option to filter this list

```

Working with Logs and Log Publishers

UnboundID supports different types of log publishers that can be used to provide the monitoring information for operations, access, debug, and error messages that occur during normal server processing. The server provides default log files as well as mechanisms to configure custom log publishers with their own log rotation and retention policies.

Types of Log Publishers

Log publishers can be used to log processing information about the server, including:

- **Error loggers** – provide information about warnings, errors, or significant events that occur within the server.
- **Trace logger** – provides information about each HTTP, OAuth2, XACML policy, and SCIM request and response that is processed by the Data Broker.

Viewing and Configuring Log Publishers

Log publishers can be created or modified on each server using the `dsconfig` tool or through the Management Console, **Logging, monitoring, and notifications -> Log Publishers**.

Creating a New Log Publisher

UnboundID provides customization options to create log publishers with the `dsconfig` command or through the Management Console.

After creating a new log publisher, configure the log retention and rotation policies. For more information, see [Configuring Log Rotation and Configuring Log Retention](#).

The following example shows how to create a trace logger that collects debug information for HTTP, external identity provider, XACML policy, and store adapter operations with the `dsconfig` command:

```
$ bin/dsconfig create-log-publisher \  
  --publisher-name NewTraceLogger \  
  --type file-based-trace \  
  --set enabled:true \  
  --set debug-message-type:external-identity-provider-request-and-response \  
  --set debug-message-type:http-full-request-and-response \  
  --set debug-message-type:policy-decision-trace \  
  --set debug-message-type:store-adapter-processing \  
  --set http-message-type:request \  
  --set http-message-type:response \  
  --set xacml-policy-message-type:result \  
  --set 'exclude-path-pattern:/**/*.css' \  
  --set 'exclude-path-pattern:/**/*.gif' \  
  --set 'exclude-path-pattern:/**/*.jpg' \  
  --set 'exclude-path-pattern:/**/*.png' \  
  --set log-file:myfile \  
  --set "rotation-policy:24 Hours Time Limit Rotation Policy" \  
  --set "rotation-policy:Size Limit Rotation Policy" \  
  --set "retention-policy:File Count Retention Policy" \  

```

```
--set "retention-policy:Free Disk Space Retention Policy" \
--set compression-mechanism:gzip
```

Compression cannot be disabled or turned off once configured for the logger. Determine logging requirements before configuring this option.

Configuring Log Compression

UnboundID servers support the ability to compress log files as they are written. Because of the inherent problems with mixing compressed and uncompressed data, compression can only be enabled when the logger is created. Compression cannot be turned on or off once the logger is configured. If the server encounters an existing log file at startup, it will rotate that file and begin a new one rather than attempting to append it to the previous file.

Compression is performed using the standard gzip algorithm. Because it can be useful to have an amount of uncompressed log data for troubleshooting, having a second logger defined that does not use compression may be desired.

Configure compression by setting the `compression-mechanism` property to have the value of `gzip` when creating a new logger. See [Creating a New Log Publisher](#) for details.

Configuring Log Signing

UnboundID servers support the ability to cryptographically sign a log to ensure that it has not been modified. For example, financial institutions require tamper-proof audit logs files to ensure that transactions can be properly validated and ensure that they have not been modified by a third-party entity or internally by an unauthorized person.

When enabling signing for a logger that already exists, the first log file will not be completely verifiable because it still contains unsigned content from before signing was enabled. Only log files whose entire content was written with signing enabled will be considered completely valid. For the same reason, if a log file is still open for writing, then signature validation will not indicate that the log is completely valid because the log will not include the necessary "end signed content" indicator at the end of the file.

To validate log file signatures, use the `validate-file-signature` tool provided in the `bin` directory of the server (or the `bat` directory on Windows systems). Once this property is enabled, disable and then re-enable the log publisher for the changes to take effect. Perform the following steps to configure log signing:

1. Use `dsconfig` to enable log signing for a Log Publisher. In this example, set the `sign-log` property on the File-based Trace Log Publisher.

```
$ bin/dsconfig set-log-publisher-prop \
--publisher-name "File-Based Trace Logger" \
--set sign-log:true
```

2. Disable and then re-enable the Log Publisher for the change to take effect.

```
$ bin/dsconfig set-log-publisher-prop \
--publisher-name "File-Based Trace Logger" \
--set enabled:false
```

Chapter 9: Advanced Configuration

```
$ bin/dsconfig set-log-publisher-prop \  
--publisher-name "File-Based Trace Logger" \  
--set enabled:true
```

3. To validate a signed file, use the `validate-file-signature` tool to check if a signed file has been altered.

```
$ bin/validate-file-signature --file logs/trace
```

```
All signature information in file 'logs/trace' is valid
```

If any validation errors occur, a message displays that is similar to this:

```
One or more signature validation errors were encountered while  
validating the contents of file 'logs/trace':  
* The end of the input stream was encountered without encountering the  
end of an active signature block. The contents of this signed block  
cannot be trusted because the signature cannot be verified
```

Configuring Log Retention and Log Rotation Policies

UnboundID servers enable configuring log rotation and log retention policies.

Log Retention – When any retention limit is reached, the server removes the oldest archived log prior to creating a new log. Log retention is only effective if a log rotation policy is in place. A new log publisher must have at least one log retention policy configured. The following policies are available:

- **File Count Retention Policy** – Sets the number of log files you want the server to retain. The default file count is 10 logs. If the file count is set to 1, the log will continue to grow indefinitely without being rotated.
- **Free Disk Space Retention Policy** – Sets the minimum amount of free disk space. The default free disk space is 500 MB.
- **Size Limit Retention Policy** – Sets the maximum size of the combined archived logs. The default size limit is 500 MB.
- **Custom Retention Policy** – Create a new retention policy that meets the server's requirements.
- **Never Delete Retention Policy** – Used in a rare event that does not require log deletion.

Log Rotation – When a rotation limit is reached, the server rotates the current log and starts a new log. A new log publisher must have at least one log rotation policy configured. The following policies are available:

- **Time Limit Rotation Policy** – Rotates the log based on the length of time since the last rotation. Default implementations are provided for rotation every 24 hours and every seven days.

- **Fixed Time Rotation Policy** – Rotates the logs every day at a specified time (based on 24-hour). The default time is 2359.
- **Size Limit Rotation Policy** – Rotates the logs when the file reaches the maximum size. The default size limit is 100 MB.
- **Never Rotate Policy** – Used in a rare event that does not require log rotation.

Configure the Log Rotation Policy

Use `dsconfig` to modify the log rotation policy for the access logger:

```
$ bin/dsconfig set-log-publisher-prop \
--publisher-name "File-Based Error Logger" \
--remove "rotation-policy:24 Hours Time Limit Rotation Policy" \
--add "rotation-policy:7 Days Time Limit Rotation Policy"
```

Configure the Log Retention Policy

Use `dsconfig` to modify the log retention policy for the access logger:

```
$ bin/dsconfig set-log-publisher-prop \
--publisher-name "File-Based Error Logger" \
--set "retention-policy:Free Disk Space Retention Policy"
```

Monitoring the Server

While the server is running, it generates a significant amount of information available through monitor entries. This section contains information about the following:

- [Backend Monitor Entries](#)
- [Viewing System and Consent Data through the Metrics Engine](#)
- [Using the status Tool](#)

Backend Monitor Entries

Each UnboundID server exposes its monitoring information under the `cn=monitor` entry. Administrators can use various means to monitor the servers through SNMP, LDAP command-line tools, and the Stats Logger.

The Monitor Backend contains an entry per component or activity being monitored. The list of all monitor entries can be seen using the `ldapsearch` command as follows:

```
$ bin/ldapsearch --hostname server1.example.com \
--port 1389 \
--bindDN "uid=admin,dc=example,dc=com" \
--bindPassword secret \
--baseDN "cn=monitor" "(objectclass=*)" cn
```

The following table lists a subset of monitor entries.

Monitoring Components

Component	Description
Active Operations	Provides information about the operations currently being processed by the server including the number of operations, information on each operation, and the number of active persistent searches.
Backends	Provides general information about the state of a server backend, including the entry count. If the backend is a local database, there is a corresponding database environment monitor entry with information on cache usage and on-disk size.
Client Connections	Provides information about all client connections to the server including a name followed by an equal sign and a quoted value, such as <code>connID="15"</code> , <code>connectTime="20100308223038Z"</code> .
Connection Handlers	Provides information about the available connection handlers on the server including the LDAP and LDIF connection handlers.
Disk Space Usage	Provides information about the disk space available to various components of the server.
General	Provides general information about the state of the server, including product name, vendor name, and server version.
Index	Provides information on each index including the number of preloaded keys and counters for read, write, remove, open-cursor, and read-for-search actions. These counters provide insight into how useful an index is for a given workload.
HTTP/HTTPS Connection Handler Statistics	Provides statistics about the interaction that the associated HTTP connection handler has had with its clients, including the number of connections accepted, average requests per connection, average connection duration, total bytes returned, and average processing time by status code.
JVM Stack Trace	Provides a stack trace of all threads processing within the JVM.
LDAP Connection Handler Statistics	Provides statistics about the interaction that the associated LDAP connection handler has had with its clients, including the number of connections established and closed, bytes read and written, LDAP messages read and written, and operations initiated, completed, and abandoned.
Processing Time Histogram	Categorizes operation processing times into a number of user-defined buckets of information, including the total number of operations processed, overall average response time (ms), and number of processing times between 0ms and 1ms.
System Information	Provides general information about the system and the JVM on which the server is running, including system host name, operation system, JVM architecture, Java home, and Java version.
Version	Provides information about the server version, including build ID, and revision number.
Work Queue	<p>Provides information about the state of the server work queue, which holds requests until they can be processed by a worker thread, including the requests rejected, current work queue size, number of worker threads, and number of busy worker threads.</p> <p>The work queue configuration has a <code>monitor-queue-time</code> property set to <code>true</code> by default. This logs messages for new operations with a <code>qtime</code> attribute included in the log messages. Its value is expressed in milliseconds and represents the length of time that operations are held in the work queue.</p>

Viewing System and Consent data Through the Metrics Engine

The Metrics Engine contains several charts to measure and monitor Data Broker system and user consent activity. Charts and data are configured from the Metrics Engine Server. The following categories can be made available through a Metrics Engine dashboard:

Authorization Requests – Displays the number of blocked and permitted token requests from client applications.

Request Volume – Displays authorization activity according to grant or deny.

Grant Types – Displays the number of authorization grants by type.

Consent/Deny by Application – Displays authorization activity based on client application.

Consent/Deny by Data Type – Displays authorization activity based on data type.

Most Requested Data – Displays most requested data.

Most Active Applications – Displays most active client applications.

Most Active Policies – Displays most active policies.

See the *UnboundID Metrics Engine Administration Guide* for more information.

Using the status Tool

UnboundID servers provide the `status` tool, which lists the health of the server. The `status` tool polls the current health of the server and displays summary information about the number of operations processed in the network. The tool provides the following information:

Status Tool Sections

Status Section	Description
Server Status	Displays the server start time, operation status, number of connections (open, max, and total).
Server Details	Displays the server details including host name, administrative users, install path, server version, and Java version.
Connection Handlers	Displays the state of the connection handlers including address, port, protocol and current state.
Admin Alerts	Displays the 15 administrative alerts that were generated over the last 48-hour period. Limit the number of displayed alerts using the <code>--maxAlerts</code> option. For example, <code>status --maxAlerts 0</code> suppresses all alerts.

Server SDK Extensions

Custom server extensions can be created with the UnboundID Server SDK. Extension bundles are installed from a .zip archive or a file system directory. Use the `manage-extension` tool to install or update any extension that is packaged using the extension bundle format. It opens and loads the extension bundle, confirms the correct extension to install, stops the server if necessary, copies the bundle to the server install root, and then restarts the server.

Note

The `manage-extension` tool must be used with Java extensions packaged using the

extension bundle format. For more information, see the "Building and Deploying Java-Based Extensions" section of the Server SDK documentation.

The UnboundID Server SDK enables creating extensions for the Data Store, Proxy, Metrics Engine, Data Broker, and Data Sync servers. Cross-product extensions include:

- Access Loggers
- Alert Handlers
- Error Loggers
- Key Manager Providers
- Monitor Providers
- Trust Manager Providers
- OAuth Token Handlers
- Manage Extension Plugins

Extensions for the Data Broker include:

- Policy Information Provider
- Store Adapter

Data Broker Advanced Server Configuration

When a Data Broker is set up from a peer, its server configuration is cloned to the new Data Broker, and the two configurations are linked such that changes to the configuration are applied to both Data Broker servers by default. See [Installing a Clone Data Broker](#). If a server is installed in an existing topology (an installation option), the server configurations are also linked.

The server's configuration is stored in an LDIF-based backend under the `cn=config` base DN. It can be accessed using the LDAP protocol and is managed by the `dsconfig` tool, Configuration API, or the Data Broker Console.

Configuring Third-Party Store Adapters

Third-party adapters can be created for data stores, that are not the UnboundID Data Store, with the Server SDK available in the `unboundid-server-sdk-<version>.zip` package.

Configuring a custom store adapter includes the following steps:

1. Create a store adapter.
2. Store it in the `/extensions` directory of the Data Broker.
3. Create a SCIM Resource Type schema.
4. Map Store Adapter(s) and SCIM Resource Types using the Management Console or `dsconfig` tool.

Example Third-Party Store Adapter

The Server SDK provides an example implementation of a third-party store adapter. View the example and associated Javadocs in the Server SDK `docs/example-html/ExampleStoreAdapter.java.html` directory.

`ExampleStoreAdapter.java` is an implementation of a flat-file JSON store adapter, which stores the SCIM user data in JSON. At startup, all resources are loaded from the `json-file-path` parameter (`resource/user-database.json`). The example uses an in-memory hash map of SCIM resources mapped to their SCIM ID.

The example provides full operations plus filterable search support for add, update, and deletes. The example will perform a full-file rewrite on every change, because the file format is a serialized list of `Resources<BaseResource>`. The code example does not support sorting or resource versioning.

About Cross-Origin Resource Sharing Support

Cross-Origin Resource Sharing (CORS) enables client applications to make JavaScript requests to the Data Broker (or Data Store) by specifying the domain from which the request is made. These cross-domain requests are generally not allowed by web browsers without CORS support. CORS defines a way in which the browser and the server can interact to determine whether a request is coming from a trusted domain.

CORS Implementation

CORS is implemented per HTTP servlet extension. Access is governed by HTTP Servlet Cross Origin Policies defined through the `dsconfig` tool. Trusted domains can be added to these policies or defined with registered applications in the Data Broker Console or through the `broker-admin` tool.

Note

By default, HTTP servlet extensions do not have CORS defined. Without a CORS policy defined, the configuration of the browser will determine application access.

The following are configuration options in `dsconfig`:

```
>>>> HTTP Servlet Cross Origin Policy management menu

What would you like to do?

1) List existing HTTP Servlet Cross Origin Policies
2) Create a new HTTP Servlet Cross Origin Policy
3) View and edit an existing HTTP Servlet Cross Origin Policy
4) Delete an existing HTTP Servlet Cross Origin Policy

b) back
q) quit

Enter option [b]:
```

HTTP Servlet Services

Enabling CORS for a particular servlet can impact another service provided by the same servlet. It is important to know which services will be affected when enabling CORS for an Data Broker servlet. The following are available servlets and their functions.

Servlet	Functions
API Explorer Servlet	Manages requests to the API Explorer, which enables testing Data Broker functions.
Configuration	Used to enable read and write access to the server's Configuration API.
Documentation	Manages requests for the <code>/docs</code> content, which includes the <code>index.html</code> page, the generated Configuration Reference Guide, and other product documents.
OAuth Servlet	OAuth2 authorization, token, revocation, and validation endpoints.
Policy Decision Point Servlet	XACML PDP endpoint.
SCIM2	Profile access by SCIM Resource Type using SCIM.
Spring Security	Authentication and authorization layer for the rest of the servlets. Data Broker login and registration endpoints.
UserInfo Servlet	Profile access using OpenID Connect.
Velocity	Velocity templates, including the Data Broker's login, registration, and consent interfaces.

Note

Any servlet accepting JavaScript calls from client applications that are hosted at a different location than that of the Data Broker APIs, such as the Velocity servlet, must have CORS enabled.

HTTP Servlet Cross Origin Policies

Two sample policies are available after installation. They can be associated with a servlet extension, or used as templates for additional policies.

Per-Application Origins – This policy trusts origins that are listed as trusted by applications registered with the Data Broker.

Restrictive – This policy rejects all cross-origin requests unless explicitly defined with the `cors-allowed-origins` property. Requests from application origins that are not specified are rejected with a 403 `Forbidden` return code.

Each policy accepts values for the following properties.

Property	Description
<code>cors-enabled</code>	Specifies if the CORS protocol is allowed by the servlet. The default value is <code>false</code> .
<code>cors-allowed-methods</code>	Specifies the list of HTTP methods allowed for access to resources. The default value is <code>GET</code> .
<code>cors-enable-per-application-origins</code>	Specifies that a per-application list of allowed origins is consulted. The default value is <code>false</code> in the Restrictive policy and <code>true</code> in the Per-Application Origins policy.

Property	Description
<code>cors-allowed-origins</code>	Specifies a global list of allowed origins. If the <code>cors-enable-per-application-origins</code> property is set to <code>true</code> , and there are origins listed here, this list is consulted in addition to the per-application list. A value of "*" specifies that all origins are allowed. The default is an empty list.
<code>cors-exposed-headers</code>	Specifies a list of HTTP headers that browsers are allowed to access. Simple response headers, as defined in the Cross-Origin Resource Sharing Specification, are allowed. The default is an empty list.
<code>cors-allowed-headers</code>	Specifies the list of header field names that are supported for a resource and can be specified in a cross-origin request. The default values are <code>Origin</code> , <code>Accept</code> , <code>X-Requested-With</code> , <code>Content-Type</code> , <code>Access-Control-Request-Method</code> , and <code>Access-Control-Request-Headers</code> .
<code>cors-preflight-max-age</code>	Specifies the maximum number of seconds that a preflight request can be cached by the client. The default value is <code>1800</code> (30 minutes).
<code>cors-allow-credentials</code>	Specifies whether requests that include credentials are allowed. This value should be <code>false</code> for servlets that use OAuth2 authorization. The default value is <code>false</code> .

Assigning a CORS Policy to an HTTP Servlet Extension

CORS policies are assigned to HTTP servlet extensions through `dsconfig`.

The following are configuration options for the SCIM servlet extension:

```
>>>> Configure the properties of the SCIM Resource Type SCIM HTTP Servlet Extension
Property          Value(s)
-----
1) description      -
2) cross-origin-policy  No cross-origin policy is defined and no CORS headers are
   recognized or returned.
3) base-context-path  /scim

?) help
f) finish - apply any changes to the SCIM Resource Type SCIM HTTP Servlet Extension
a) show advanced properties of the SCIM Resource Type SCIM HTTP Servlet Extension
d) display the equivalent dsconfig command lines to either re-create this object or only
   to apply pending changes
b) back
q) quit

Enter option [b]: 2
```

Choose the `cross-origin-policy` option. Defined policies are listed.

```
>>>> Configuring the 'cross-origin-policy' property
The cross-origin request policy to use for the HTTP Servlet Extension.

A cross-origin policy is a group of attributes defining the level of cross-origin request
supported by the HTTP Servlet Extension.
```

Chapter 9: Advanced Configuration

Do you want to modify the 'cross-origin-policy' property?

- 1) Keep the default behavior: No cross-origin policy is defined and no CORS headers are recognized or returned.
 - 2) Change it to the HTTP Servlet Cross Origin Policy: Per-Application Origins
 - 3) Change it to the HTTP Servlet Cross Origin Policy: Restrictive
 - 4) Create a new HTTP Servlet Cross Origin Policy
- ?) help
q) quit

Choose the CORS policy to assign to this servlet extension.

Managing Server Encryption Settings

The server encryption settings database is managed by the `encryption-settings` command-line tool. The keys stored for the server are used to encrypt tokens, authorization codes, account linking codes, and external identity provider tokens. Encryption settings definitions can be created, listed, exported and imported. Help and examples are available with the following command:

```
$ bin/encryption-settings --help
```

Information about the cipher algorithms and transformations available for use is located in the *Java Cryptography Architecture Reference Guide* and *Standard Algorithm Name Documentation* available on the Oracle website.

Rotating the Encryption Key

Perform the following steps for routine rotation of the encryption key:

1. Create a new encryption settings definition.

```
$ encryption-settings create \  
  --cipher-algorithm AES \  
  --key-length-bits 128
```

```
Successfully created a new encryption settings definition with ID <ID>
```

2. Verify the new definition was created.

```
$ encryption-settings list  
Encryption Settings Definition ID: <old-key>  
  Preferred for New Encryption: true  
  Cipher Transformation: AES  
  Key Length (bits): 128  
  
Encryption Settings Definition ID: <ID>  
  Preferred for New Encryption: false  
  Cipher Transformation: AES  
  Key Length (bits): 128
```

3. Create a PIN file that will be used for the exported definition.

```
$ echo "secret" > /tmp/exported-key.pin
```


4. Export the encrypt settings, referring to the generated encryption settings ID.

```
$ encryption-settings export \
  --id <ID> \
  --output-file /tmp/exported-key \
  --pin-file /tmp/exported-key.pin

Successfully exported encryption settings definition <ID> to file
/tmp/exported-key
```

5. For every Data Broker instance in the topology, copy the exported definition and PIN file to the Data Broker's host. Import the encryption settings, without setting them as preferred. Delete the exported settings and PIN file when finished.

```
$ encryption-settings import \
  --input-file /tmp/exported-key \
  --pin-file /tmp/exported-key.pin

Successfully imported encryption settings definition <ID> from file
/tmp/exported-key

$ rm /tmp/exported-key
$ rm /tmp/exported-key.pin
```

6. After importing the encryption settings definition to all Brokers, for each one, including the Broker where the definition was originally created, set the new definition as preferred.

```
$ encryption-settings set-preferred \
  --id <ID>

Encryption settings definition <ID> is was successfully set as the
preferred definition for subsequent encryption operations.
```

Addressing a Compromised Encryption Key

If an encryption settings definition becomes compromised, perform the following to create a new definition and update the Data Broker servers. See the command line help for the `encryption-settings` tool for arguments.

Note

If the Data Broker's encryption key is compromised, and the Broker has been collecting access tokens for external identity providers through the relying party feature, make sure those tokens are revoked.

1. Back up the encryption settings backend.
2. Back up the user store.
3. Revoke all authorizations for each client.
4. Stop the HTTPS Connection Handler that is used for the Data Broker's REST APIs.

Chapter 9: Advanced Configuration

```
$ dsconfig set-connection-handler-prop \  
  --handler-name "HTTPS Connection Handler" \  
  --set enabled:false
```

5. Create a new encryption settings definition and set it as preferred. The following will encrypt data using a 128-bit AES cipher:

```
$ encryption-settings create \  
  --cipher-algorithm AES \  
  --key-length-bits 128 \  
  --set-preferred
```

6. Restart the HTTPS Connection Handler.

```
$ dsconfig set-connection-handler-prop \  
  --handler-name "HTTPS Connection Handler" \  
  --set enabled:true
```

If the deployment includes multiple Data Brokers, all servers should be taken offline, and the encryption settings database must be updated on every server.

Note

Do not delete the compromised encryption definition. It will still be used to decrypt tokens, authorization codes, and links that were encrypted with the previous key.

Account Recovery Configuration in the Data Store

End users can recover Data Broker account information or reset a password, if a Data Store is configured as the primary User Store and one-time passwords (OTP) are enabled. See the *UnboundID Data Store Administration Guide* for details about configuring one-time passwords.

On the Data Broker server, configuration for account recovery and new account registration are enabled by configuring the [Identity Provider Service](#) through the Management Console or with the `dsconfig` tool.

Configuration on the Data Store requires creating and enabling the OTP mechanism and defining the delivery mechanism for reset tokens, as follows:

1. Create and enable the OTP delivery mechanism. The following uses the sample user starter schema that is enabled for the sample applications.

```
$ bin/dsconfig create-otp-delivery-mechanism \  
  --mechanism-name "Email OTP Delivery Mechanism" \  
  --type email \  
  --set enabled:true \  
  --set email-address-attribute-type:ubidemailjson \  
  --set email-address-json-field:value \  
  --set 'email-address-json-object-filter:  
    (ubidemailjson:jsonObjectFilterExtensibleMatch:={  
"filterType":"equals",  
  "field":"verified", "value":true })' \  
  --set 'sender-address:do-not-reply@example.com'
```

2. Define the email server to deliver reset tokens:

```
$ bin/dsconfig create-external-server \
  --server-name "Example.com SMTP" \
  --type smtp \
  --set server-host-name:smtp.example.com

$ bin/dsconfig set-global-configuration-prop \
  --set "smtp-server:Example.com SMTP"
```

3. Create and enable the extended operations handlers to generate and send reset tokens:

```
$ bin/dsconfig create-extended-operation-handler \
  --handler-name "Single Use Tokens" \
  --type single-use-tokens \
  --set enabled:true \
  --set "password-generator:One-Time Password Generator" \
  --set "default-otp-delivery-mechanism:Email OTP Delivery Mechanism"

$ bin/dsconfig create-extended-operation-handler \
  --handler-name "Deliver Password Reset Token" \
  --type deliver-password-reset-token \
  --set enabled:true \
  --set "password-generator:One-Time Password Generator" \
  --set "default-token-delivery-mechanism:Email OTP Delivery Mechanism"
```

Configuring the Data Broker Templates

The Data Broker exposes several Velocity pages through an HTTP Servlet Extension. The pages are for login, for OAuth consent, account recovery and registration, and an error page that can be surfaced for end users and are located in:

<server-root>/config/velocity/templates

See [Configuring the Broker Login and Consent Pages](#) for information about these files.

To enable Velocity support, add the Velocity HTTP Servlet Extension to an enabled HTTP or HTTPS connection handler:

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --add http-servlet-extension:Velocity
```

Velocity template files contain presentation content and variables that are replaced when the content is requested. Variables are expressed using a \$ followed by an identifier that refers to an object put into a context (VelocityContext) by the server.

Velocity extensions can be configured to expose a number of objects in the context using the `expose-*` properties:

- **expose-request-attributes** – Indicates whether HTTP request attributes are accessible to templates using the `$ubid_request` variable. In general, request attributes are added by server components processing the HTTP request. Also the HTTP request parameters

map is available as `$subid_request.parameters`. Request parameters are supplied by the requester, usually in the request URL query string or in the body of the request itself.

- **expose-session-attributes** – Indicates whether HTTP session attributes are accessible to templates using the `$subid_session` variable. Like request attributes, session attributes are also added by server components processing the HTTP request. The lifetime of these attributes persists until the user's session has ended.
- **expose-server-context** – Indicates whether a Server SDK server context is accessible to templates using the `$subid_server` variable. The server context provides access to properties and additional information about the server. See the *Unbound ID Server SDK* documentation for more details.

The following are other properties of the Velocity HTTP Servlet Extension:

- **base-context-path** – URL base context for the Velocity Servlet.
- **static-content-directory** – In addition to templates, the Velocity Servlet will serve miscellaneous static content related to the templates. This property defines the directory where these resources are found.
- **static-context-path** – URL path beneath the base context where static content can be accessed.
- **mime-types-file** – Specifies a file that is used to map file extensions of static content to a Content Type to be returned with requests.
- **default-mime-type** – The default Content Type for HTTP responses. Additional content types are supported by defining on or more additional Velocity Template Loaders.
- **template-directory** – The directory where templates are stored. This directory also serves as a default for Template Loaders that do not have a template directory specified explicitly.

The VelocityContext object can be further customized by configuring additional Velocity context providers. The dot notation used for context references can be extended arbitrarily to access properties and methods of objects in context using Java Bean semantics. For example, if the HTTP request URL includes a `name` query string parameter like:

```
http://example.com:8080/view/hello?name=Karl&name=Vladimir+Ilyich&name=Steve
```

An HTML template like the following could be used to generate a page containing a friendly greeting to the end user:

```
<html>
  <body>
    Hello, $subid_request.parameters.name[0], $subid_request.parameters.name[1], and
    $subid_request.parameters.name[2]!
  </body>
</html>
```

Note

For security, all template substitutions are HTML escaped by default. To substitute unescaped

content, a variable name ending with "WithHtml" must be used. For example, `$addressWithHtml`, would substitute the contents of the `$addressWithHtml` variable into the page generated from the HTML template without escaping it.

By default, the Velocity Servlet Extension expects to access content in subdirectories of the server's `config/velocity` directory:

- **templates** – This directory contains Velocity template files that are used to generate pages in response to client requests.
- **statics** – This directory contains static content such as CSS, HTML, and JavaScript files as well as images and third-party libraries.

Supporting Multiple Content Types

By default, the Velocity Servlet Extension is configured to respond to HTTP requests with a content type `text/html`. Change this request type by setting the default MIME type using `dsconfig`. For example, the following can be used to set the default type to XML:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name Velocity \
  --set default-mime-type:application/xml
```

HTML requests can be supported as well as clients that seek content in other formats. Create one or more Velocity Template Loaders to load templates for other content types like XML or JSON.

The ability to serve multiple formats of a document to clients at the same URL is typically called *content negotiation*. HTTP clients indicate the type of content desired using the `Accept` header. A client may use a header like the following to indicate that they prefer content in XML but will fallback to HTML if necessary:

```
Accept: application/xml,text/html;q=0.9
```

The following can be used to create a Velocity Template Loader for XML content:

```
$ bin/dsconfig create-velocity-template-loader \
  --extension-name Velocity \
  --loader-name XML \
  --set evaluation-order-index:502 \
  --set mime-type-matcher:application/xml \
  --set mime-type:application/xml \
  --set template-suffix:.vm.xml
```

Upon receiving a request, the Velocity Servlet first creates an ordered list of requested media types from most desired to least based on the value of the `Accept` header. Starting from the most desired type, it will then iterate over the defined Template Loaders according to the `evaluation-order-index` property from lowest value to highest.

A Template Loader may indicate that it can handle content for requested media type by comparing the requested type to its `mime-type-matcher` property. A loader may be configured to load templates from a specific directory or load template files having a particular suffix. In this case, where XML templates are expected to be named using a `.vm.xml` suffix. If a loader indicates it handles the requested content type and a template exists for the requested view,

the template is loaded and used to generate a response to the client. If no loaders are found for the requested media type, the next most preferred media type (if any) is tried. If no loaders indicated that they could satisfy the requested view, the client is sent an HTTP 404 (not found) error. If no loaders could provide acceptable media but the requested view exists in some other format, the client is sent an HTTP 406 (not acceptable) error.

In this example, a template file called `hello.vm.xml` can be used to generate a response in XML:

```
<hello name="$_request.parameters.name"/>
```

In this case, the response will contain an HTTP Content-Type header with the value of the `mime-type` property of the Velocity Template Loader.

Velocity Context Providers

The previous examples make use of value supplied as an HTTP request query string parameter to form a response. The templates contain a variable `$_request.parameters.name` that was replaced at runtime with a value from the Velocity Context.

The Velocity Extension can be configured to make some information available in the Velocity Context such as the HTTP request, session, and Server SDK Server Context. Velocity Context Providers provide more flexibility in populating the Velocity Context for template use.

Here are some of the properties of a Velocity Context Provider:

- **enabled** – Indicates whether the provider will contribute content for any requests.
- **object-scope** – Indicates to the provider how often objects contributed to the Velocity Context should be re-initialized. Possible values are: `request`, `session`, or `application`.
- **included-view/excluded-view** – These properties can be used to restrict the views for which a provider contributes content. A view name is the request URL's path to the resource without the Velocity Servlet's context or a leading forward slash. If one or more views are included, the provider will service requests for just the specified views. If one or more views are excluded, the provider will service requests for all but the excluded views.

Note

If the scope of the Velocity Tools context provider is constrained by setting the `included-view` property, the OAuth2 consent flow may be affected. The `included-view` property should not be changed, unless all system default templates are included when setting the property.

Configuring HTTP Header Fields

By default, the Velocity Extension returns a set of standard HTTP header fields in every request served by the extension, including those for directing cache policies of user agents and frame-hosting options. These header fields can be configured in the following ways:

- The Velocity Servlet Extension's response-header configuration property can be specified to add a request header to every template request. The `static-response-header` property can be specified to add a header field to requests for static content like images and script files.
- Header fields for individual pages can be configured by using the `response-header` property of the Velocity Context Provider objects, which will add the header fields to just those pages served by the provider. Headers specified here will overwrite those specified by the Velocity Extension.
- Header fields can be manipulated directly by third-party Velocity Context Providers in code, adding or removing existing headers by manipulating the HTTP servlet response directly.

Handling Specific HTTP Methods in Third-Party Providers

In addition to contributing content to the Velocity Context, Velocity Context Providers can perform actions in response to particular HTTP methods. For example, a template can be used to POST a form of user data to a provider, which in turn would create a user in the User Store. In addition to handling HTTP GET and POST operations, a provider can handle any number of the standard HTTP methods (PUT, PATCH, DELETE, HEAD). Handling these methods is a two step process.

1. When creating a third-party Velocity Context Provider, configure the HTTP methods the provider will handle using the `http-method` property. For example, the following command might be used to configure a provider to handle GET and POST requests:

```
$ dsconfig create-velocity-context-provider \
  --extension-name Velocity \
  --provider-name "My Provider" \
  --type third-party \
  --set http-method:GET \
  --set http-method:POST \
  --set extension-class:com.example.MyProvider
```

Or update an existing provider:

```
$ dsconfig set-velocity-context-provider-prop \
  --extension-name Velocity \
  --provider-name "My Provider" \
  --add http-method:POST
```

2. When implementing the Java class, override the `handlePost()` method adding code for handling POST operations. For example, use the internal `ServerContext` object to establish a connection to the server and create a new user using form data from a POST operation. Logic related to updating the context for the response may be implemented directly in the `handle<XXX>()` method or in the `updateContext()` method, which is called immediately after the relevant `handle<XXX>()` method.

Velocity Tools Context Provider

Apache's Velocity Tools project is focused on providing utility classes useful in template development. The Velocity Context can be configured by specifying Velocity Tool classes to be automatically added to the Velocity Context for template development. For more information about the Velocity Tools project, see the online product documentation.

The following command can be used to list the set of Velocity Tools that are included in the Velocity Context for general use by templates:

```
$ bin/dsconfig get-velocity-context-provider-prop \  
  --extension-name Velocity \  
  --provider-name "Velocity Tools" \  
  --property request-tool \  
  --property session-tool \  
  --property application-tool
```

Configuring the Broker Login and Consent Pages

The Data Broker exposes Velocity pages through an HTTP Servlet Extension. Templates are located in:

```
/<server-root>/config/velocity/templates
```

Account registration and password recovery require server configuration. See [User Account Registration and Recovery](#).

login.vm – Defines the Data Broker log in page and includes icons for external identity provider login. A registration form is also provided for users to create an account through the external identity provider login.

approve.vm – Defines the OAuth approval page presented to end users who need to approve access to resources. This file resides in the `/templates/oauth` directory.

error.vm – Defines the presentation of error messages displayed to end users if there is a problem with the login or consent. This file resides in the `/templates/oauth` directory.

recover-password.vm – Defines the prompt for information to search for a user account so the password can be changed. This file resides in the `/templates/account` directory.

recover-password-verify.vm – Defines the prompt for the password change code sent by the Data Broker and the new password. This file resides in the account directory of the `/templates` directory.

recover-password-success.vm – Defines the password change success notification. This file resides in the `/templates/account` directory.

recover-username.vm – Defines the prompt for information to search for an account username. This file resides in the `/templates/account` directory.

recover-username-verify.vm – Defines the prompt for the username recover code sent by the Data Broker. This file resides in the `/templates/account` directory.

recover-username-success.vm – Displays the account username. This file resides in the `/templates/account` directory.

register.vm – Provides a form for creating a new user account. This file resides in the `/templates/account` directory.

register-success.vm – Defines the notification that the user account was successfully created. This file resides in the `/templates/account` directory.

If more than one template modification is needed, additional data can be added by adjusting or adding to the context objects that are present. See [About Velocity Templates](#) for general information about adding context. Use the following two context objects when customizing these pages:

```
$principal
$requestContext
```

The \$principal Object

The OAuth2 consent page header displays the currently logged in user's name as `$principal.getObjectNode().get("userName").textValue()`. This is the placeholder variable used in the Velocity template. The Velocity template can be changed to display other attributes of the `$principal` object.

All of the SCIM principal attributes can be referenced by OAuth2 templates using SCIM's standard attribute notation. The following are examples for a SCIM object exposed as `principal` in the Velocity context.

Retrieving a simple attribute value:

```
$principal.getObjectNode().get("userName").textValue()
$principal.urn:ietf:params:scim:api:messages:2.0:employeeNumber
```

Retrieving a complex attribute value:

```
$principal.name.givenName
$principal.urn:ietf:params:scim:api:messages:2.0:manager.managerId
```

Retrieving multi-valued attribute values:

```
#foreach ( $email in $principal.emails )
$email.type: $email.value
#end
```

The \$requestContext Object

This is the context placeholder for request-specific state, such as the current web application context, the current locale, or the current theme. The following are examples of `requestContext` in the Velocity context.

Retrieving the locale of the request:

```
$requestContext.locale
```

Retrieving a Spring model object called 'token':

```
$requestContext.getModelObject('token')
```

Customizing the Data Broker Application Logo

The Data Broker's pages, can be changed or re-branded with a company logo. The application uses a cascading style sheet to determine appearance. The default style sheet file can be overwritten by creating a new style sheet for the Management Console with the following naming convention:

```
$HOME/.broker-console/branding-override.css
```

If this file is present, the Data Broker uses it to overwrite the existing style sheet.

The following is an example of the style sheet used to display the default logo in the title bar:

```
.product-logo {  
  width: 18px;  
  height: 24px;  
  background-image: url("../img/unboundid-u30.png");  
  background-size: 100% 100%;  
}
```

Style changes take affect after the application is restarted.

To Customize the Logo

By default, the web applications look for the following branding override CSS file:

```
~/.broker-console/branding-override.css
```

where "~" is replaced for the home directory of the account the web server/application is running under. It is also possible to override this file name and location by setting the "branding.override.file" System Property. If this file is found, it is included after all of the other CSS files, so that it can override any of the application's styles.

The following is an example of the CSS used to display the default logo in the title bar:

```
.product-logo {  
  width: 18px;  
  height: 24px;  
  background-image: url("../img/unboundid-u30.png");  
  background-size: 100% 100%;  
}
```

A branding-override.css file at ~/.broker-console with the following contents will display a new logo after (restart the application after creating the file):

```
.product-logo {  
  width: 550px;  
  height: 190px;  
  background-image: url(https://www.google.com/images/srpr/logo4w.png);  
  background-size: 100% 100%;  
}
```

Configuring Web Applications for Localization

To localize the Data Broker web pages, create a set of resource bundles, for each language. Locale-specific data must be tailored according to the conventions of the language and region, and isolated into locale-specific objects in a Java `ResourceBundle`. The standard naming convention is `basename_<language1>_<country1>_<variant1>`. For example:

```
messages_en_US.properties
messages_fr_FR.properties
messages_de.properties
```

Each should have the same set of keys (for example `login_prompt`, `unknown_user`) and values, which are raw text in the appropriate language. Resource Bundles and internationalization for Java are described in the Oracle documentation.

The resource bundles are loaded from the classpath as jar files in the `/lib` or `/lib/extensions` directories, or can be loaded as properties files in the server's `/classes` directory.

A Velocity Context Provider is then created to provide access to the resource bundles. Velocity Tools provide one that selects the appropriate bundle based on the locale determined from the incoming HTTP request and provides the messages from that bundle to the Velocity template. This tool class can be found at:

```
https://velocity.apache.org/tools/devel/javadoc/org/apache/velocity/tools/generic/ResourceTool.html
```

Configure an instance of this tool and specify the name of the resource bundle family ("messages" in this example). Create another properties file to configure the Velocity Tools classes:

```
/config/velocity/ResourceToolConfig.properties
```

Add the following lines:

```
bundles=Messages
#locale=en_US This can be used to enforce a specific locale.
```

Run the following `dsconfig` command to create and configure the Velocity Context Provider:

```
$ bin/dsconfig create-velocity-context-provider \
--extension-name Velocity \
--provider-name ResourceBundleProvider \
--type velocity-tools \
--set object-scope:session \
--set included-view:/path/to/template \
--set request
tool:org.apache.velocity.tools.generic.ResourceTool;config/velocity/ResourceTool.properties
```

The `included-view` is only necessary to make the localized messages available to only a certain set of templates.

Preserving Customized Files

Any files that are customized should be copied from the `config/velocity` subdirectories to the same subdirectory of the velocity directory under the server root (`<server-root>/velocity`). The files in `config/velocity` should not be modified. They are updated when the product is updated.

By default, any file of the same name under `<server-root>/velocity` will be loaded in place of `<server-root>/config/velocity`. This enables the preservation of customized files after a product upgrade.

After a product upgrade, review the files in `config/velocity` to determine if any changes should be incorporated into customized templates.

Topology Configuration

Topology configuration enables grouping servers and mirroring configuration changes automatically. It uses a master/slave architecture for mirroring shared data across the topology. All writes and updates are forwarded to the master, which forwards them to all other servers. Reads can be served by any server in the group.

Servers can be added to an existing topology at installation. See [Adding Additional Data Brokers in a Topology](#) for details.

Note

To remove a server from the topology, it must be uninstalled with the `uninstall` tool. See [Uninstalling the Data Broker](#) for details.

Topology Master Requirements and Selection

A topology master server receives any configuration change from other servers in the topology, verifies the change, then makes the change available to all connected servers when they poll the master. The master always sends a digest of its subtree contents on each update. If the node has a different digest than the master, it knows it's not synchronized. The servers will pull the entire subtree from the master if they detect that they are not synchronized. A server may detect it is not synchronized with the master under the following conditions:

- At the end of its periodic polling interval, if a server's subtree digest differs from that of its master, then it knows it's not synchronized.
- If one or more servers have been added to or removed from the topology, the servers will not be synchronized.

The master of the topology is selected by prioritizing servers by minimum supported product version, most available, newest server version, earliest start time, and startup UUID (a smaller UUID is preferred).

After determining a master for the topology group (cluster), the topology data is reviewed from all available servers (every five seconds by default) to determine if any new information makes a server better suited to being the master. If a new server can be the master, it will communicate that to the other servers, if no other server has advertised that it should be the master. This ensures that all servers accept the same master at approximately the same time (within a few milliseconds of each other). If there is no better master, the initial master maintains the role.

After the best master has been selected for the given interval, the following conditions are confirmed:

- A majority of servers is reachable from that master. (The master server itself is considered while determining this majority.)
- There is only a single master in the entire topology.

If either of these conditions is not met, the topology is without a master and the peer polling frequency is reduced to 100 milliseconds to find a new master as quickly as possible. If there is no master in the topology for more than one minute, a `mirrored-subtree-manager-no-master-found` alarm is raised. If one of the servers in the topology is forced as master with the `force-as-master-for-mirrored-data` option in the Global Configuration configuration object, a `mirrored-subtree-manager-forced-as-master-warning` warning alarm is raised. If multiple servers have been forced as masters, then a `mirrored-subtree-manager-forced-as-master-error` critical alarm will be raised.

Topology Components

When a server is installed, it can be added to an existing topology, which will clone the server's . Topology settings are designed to operate without additional configuration. If required, some settings can be adjusted to fit the needs of the environment.

Server Configuration Settings

Configuration settings for the topology are configured in the Global Configuration and in the Config File Handler Backend. Though they are topology settings, they are unique to each server and are not mirrored. Settings must be kept the same on all servers.

The Global Configuration object contains a single topology setting, `force-as-master-for-mirrored-data`. This should be set to `true` on only one of the servers in the topology, and is used only if a situation occurs where the topology cannot determine a master because a majority of servers is not available. A server with this setting enabled will be assigned the role of master, if no suitable master can be determined. See [Topology Master Requirements and Selection](#) for details about how a master is selected for a topology.

The Config File Handler Backend defines three topology (`mirrored-subtree`) settings:

- `mirrored-subtree-peer-polling-interval` – Specifies the frequency at which the server polls its topology peers to determine if there are any changes that may warrant a new master selection. A lower value will ensure a faster failover, but it will also cause more traffic among the peers. The default value is five seconds. If no suitable master is

found, the polling frequency is adjusted to 100 milliseconds until a new master is selected.

- `mirrored-subtree-entry-update-timeout` – Specifies the maximum length of time to wait for an update operation (add, delete, modify or modify-dn) on an entry to be applied by the master on all of the servers in the topology. The default is 10 seconds. In reality, updates can take up to twice as much time as this timeout value if master selection is in progress at the time the update operation was received.
- `mirrored-subtree-search-timeout` – Specifies the maximum length of time in milliseconds to wait for search operations to complete. The default is 10 seconds.

Topology Settings

Topology meta-data is stored under the `cn=topology,cn=config` subtree and cluster data is stored under the `cn=cluster,cn=config` subtree. The only setting that can be changed is the cluster name.

Monitor Data For the Topology

Each server has a monitor that exposes that server's view of the topology in its monitor backend, so that peer servers can periodically read this information to determine if there are changes in the topology. Topology data includes the following:

- The server ID of the current master, if the master is not known.
- The instance name of the current master, or if a master is not set, a description stating why a master is not set.
- A flag indicating if this server thinks that it should be the master.
- A flag indicating if this server is the current master.
- A flag indicating if this server was forced as master.
- The total number of configured peers in the topology group.
- The peers connected to this server.
- The current availability of this server
- A flag indicating whether or not this server is not synchronized with its master, or another node in the topology if the master is unknown.
- The amount of time in milliseconds where multiple masters were detected by this server.
- The amount of time in milliseconds where no suitable server is found to act as master.
- A SHA-256 digest encoded as a base-64 string for the current subtree contents.

The following metrics are included if this server has processed any operations as master:

- The number of operations processed by this server as master.
- The number of operations processed by this server as master that were successful.

- The number of operations processed by this server as master that failed to validate.
- The number of operations processed by this server as master that failed to apply.
- The average amount of time taken (in milliseconds) by this server to process operations as the master.
- The maximum amount of time taken (in milliseconds) by this server to process an operation as the master.

Index

A

access token 51

- accepting external tokens 82
- authorization code grant 71
- client credentials code grant 75
- implicit code grant 73
- password credentials code grant 74

access token properties 112

account creation 27, 33

account operation requests 103

account operations

- scope properties 52, 109

account recovery configuration 158

account registration 28, 33

- Identity Provider settings 58

account registration template 165

account username and password recovery 27, 33

admin entitlement 55, 76

administrative account

- adding a root user account 126
- root user privileges 126

API Explorer 3, 118

attribute mappings 40

- authoritative attribute 40, 46
- complex attributes 47
- described 37
- indexing 46
- mapping in SCIM Resource Types 46
- userinfo claims 46

Authenticated Identity scope 52

authentication

- define SCIM search filters for usernames 60

authoritative attribute 40

authorization

- viewing consent metrics 151

authorization code grant request 70

B

backend monitors

- entries 149

backup tool 124

base DN

- configure data store 15
- configure user entries 18

base64 tool 124

broker-cfg.dsconfig

- write file 18

C

Claims Map

- described 37

client

- REST API endpoints 85

client-specific SCIM attributes 48

client credentials code grant request 75

- considerations 75

client identifier 63, 72, 74-75

client secret 63, 74-75

collect-support-data tool 124

config-diff tool 124

Config File Handler Backend 169

consent-admin tool 124

consent operation requests 103

correlation attribute 38

CORS

configuration 153

create-initial-broker-config 17

create-initial-broker-config tool 124

create-rc-script tool 124

D

Data Broker

architecture 3

attribute filtering 2-3

authorization 2

described 1

features 2

in a topology 19

installing 16

installing with existing truststore 22

Management Console URL 18

pluggable authentication 2

sample workflow 8

social login 3

tools 124

data stores

described 37

installing 13

DNS caching 142

dsconfig

changing policy-combining
algorithm 96

CORS configuration 153

described 125

tool described 124

dsframework tool 124

dsjavaproperties tool 124

dstat

installing on SuSE Linux 12

E

encryption-settings tool 13, 124

encryption keys 13, 51, 157

endpoints

described 37

logout.do 83

SCIM 84

SCIM examples 86

token 79

token revocation 81

token validation 80

userinfo 84

error logger 146

error message template 164

evaluate-policy tool 124

external access tokens 82

external identity operations 53, 109

external identity provider operation
requests 103

external identity providers

feature 3

G

Global Configuration object 169

H

HTTP request properties 110

HTTP Servlet Cross Origin Policy 154

HTTP servlet extension 155

I

ID token 51, 63

parameters 64

ID Token Grant requests 76

- Identity Authenticator 57
 - define settings 60
- Identity Provider Service 158
 - account recovery settings 35
- implicit code grant request 72
- installing
 - prerequisites 11
 - sample users 23
 - scripted install 21
- IP address reverse name lookup 143
- J**
- Java
 - installing the JDK 13
 - supported versions 11
- JSON 40
 - object examples 86
- JVM memory allocation
 - Data Broker 17
 - data store 15
- JWT token grant type 76
- L**
- ldapmodify tool 124
- ldappasswordmodify tool 124
- LDAPS
 - configure Data Broker 16
 - configure data store 14
- ldapsearch tool 124
- ldif-diff tool 124
- ldifmodify tool 124
- Linux configuration
 - set file descriptor limit 11
- list-backends tool 125
- localization for web applications 167

- logging
 - available log publishers 146
 - configure log retention and rotation 148
 - configure log signing 147
 - create log publisher 146
 - log compression 147
- login account 18
- login page 27, 32
- logout endpoint 83
- M**
- manage-extension tool 125
- Management Console
 - login account 18
 - URL 18
- metrics
 - viewing 151
- monitoring entries 150
- O**
- OAuth2
 - authorization code grant 69
 - client credentials 70
 - described 51
 - encryption keys 51
 - endpoints
 - REST APIs 85
 - ID token 70
 - implicit grant flow 69
 - policy processing 98
 - resource owner password flow 70
 - response types 70
 - userinfo claims mapping 46
- oauth2-request tool 125

OAuth2 clients 78

- enable client-specific SCIM attributes 48
- properties in XACML Policies 108

OpenID Connect

- about 62
- ID token 63
- requests 63
- responses 63
- userinfo endpoint 85

P

password credentials code grant request 73

password recovery 35

- Identity Provider settings 59

password reset 30

PDP endpoint 104

policy

- authorization scenarios 94
- decision trace 119
- described 93
- managing 117
- PDP endpoint 104
- policy information providers 117
- policy structure 96
- request processing 98
- test policies 118
- troubleshoot denied access 119
- viewing policy metrics 151
- XACML 94

policy set

- creating 118

prepare-external-store tool 18, 125

Profile Manager application 3

- new user registration 28
- overview 26
- user search 27

R

reCAPTCHA 30

- enable in Identity Provider Service 57

reCAPTCHA API

- Identity provider settings 60

recover account username and password 35

recover password template 164

recover username template 164

redirect URI 65

referenced SCIM resource attributes 107

refresh token

- process 81

register new account 35

relying party

- add identity provider 65
- create an account 65
- Facebook settings 66
- Google settings 66
- login template 164
- OpenID Connect settings 67
- process overview 64

remove-defunct-server tool 25, 125

resource operations

- scope properties 52, 109

resource properties 107

Resource Scope 53

- use client credential grant type 76

REST API

connection port 16

endpoints 85

restore tool 125

review-licence tool 125

root user DN 18

root user privileges 126

S

Sample Sign-In application 3, 31

SCIM

described 86

SCIM endpoint 37, 84

account operations 103

consent operations 103

DELETE operations 102

external identity provider
operations 103

GET operations 100

PATCH and PUT operations 101

POST operations 101

search considerations 41

search request 100

sub-resource operations 102

SCIM request properties 110

SCIM resource properties 107

SCIM Resource Type

creating 42

managing 42

map userinfo claims 47

REST API endpoints 85

store adapter mapping 45

SCIM schema

overview 40

username uniqueness 41

scope properties in policy 107

scopes

applicable scopes returned 111

Authenticated Identity 52

external identity operations 53, 109

managing 55

Resource scope 53

scope types and properties 51

self-service account flows 58

server-state tool 125

session properties 111

social login 64

start-broker

running in the foreground 24

start-broker tool 125

status tool 125, 151

stop-broker

example of 24

in-core restart 25

stop-broker tool 125

store adapter

correlation attribute 38

described 37

mapping attributes 40

primary and secondary adapters 38

search considerations 41

third-party store adapters 152

store adapters

described 38

third-party 153

sum-file-sizes tool 125

supported platforms 11

system entropy 12

T

Third-Party Store Adapter 153

token endpoint 63, 85

 token validation 80-81

topology

 force master setting 169

 master selection 168

 monitor data 170

 overview 168

 server configuration settings 169

 subtree polling interval 168

trace logger 146

trace policy decisions 119

U

uid 41

uninstall tool 25

user processes

 configuring on Redhat/CentOS 12

user store 13

Userinfo claims

 create maps 47

 managing 46

Userinfo endpoint 37, 51, 63, 84

 described 37

 example 91

 policy requests 104

username

 SCIM search filters for
 authentication 60

username recovery 30, 35

username recovery settings 59

V

Velocity templates 35

 configuring pages 159, 164

 HTTP header fields 162

 HTTP methods in third-party
 providers 163

X

XACML

 described 94

 request attributes 100

 unsupported features 120