Ping Identity® Data Governance Broker Administration Guide

Version 6.0.0.0

Copyright

Table of Contents

-

Preface

The PingData Data Governance 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.

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

Documentation

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

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

Chapter 1: Introduction

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

Topics include:

Data Governance Broker Overview

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

The Data Governance 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 Governance 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 Governance 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 Governance 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 Directory Servers to SCIM Resource Types defined in the Data Governance Broker.

Data Governance Broker Features

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

- **Support for multiple backend Servers**. The Data Governance Broker supports multiple directory servers, with native support for the Ping Identity Directory Server and extension points for others. Directory Servers 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 Governance Broker and receive data from any type of infrastructure backend.
- **Standards-based authentication and authorization**. The Data Governance Broker provides OAuth2 and OpenID Connect-compliant functionality for authentication with the Data Governance 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.

- **Modular authentication and authorization flows**. The Data Governance Broker enables multi-factor authentication and a modular (chain-based) configuration for account flows.
- **Authorization based on XACML policy and User Consent**. The Data Governance 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 Governance 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 Governance Broker can act as a relying party, enabling users to log into OAuth2 clients and update or create Data Governance Broker accounts with external identity provider accounts from Facebook, Google, a SAML provider, or an OpenID Connect provider.
- **User interface sample**. A My Account sample application, in the `<server-root>/samples` directory, can be installed with the Data Governance 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 Governance Broker returns that data.
- **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 Governance 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`.
- **Authentication Developer Portal**. The Developer Portal enables client application developers to work with the authentication, SCIM, and UserInfo APIs to design applications that can access Data Governance Broker resources. See the Authentication Developer Portal for configuration examples (`https://developer.unboundid.com/`).

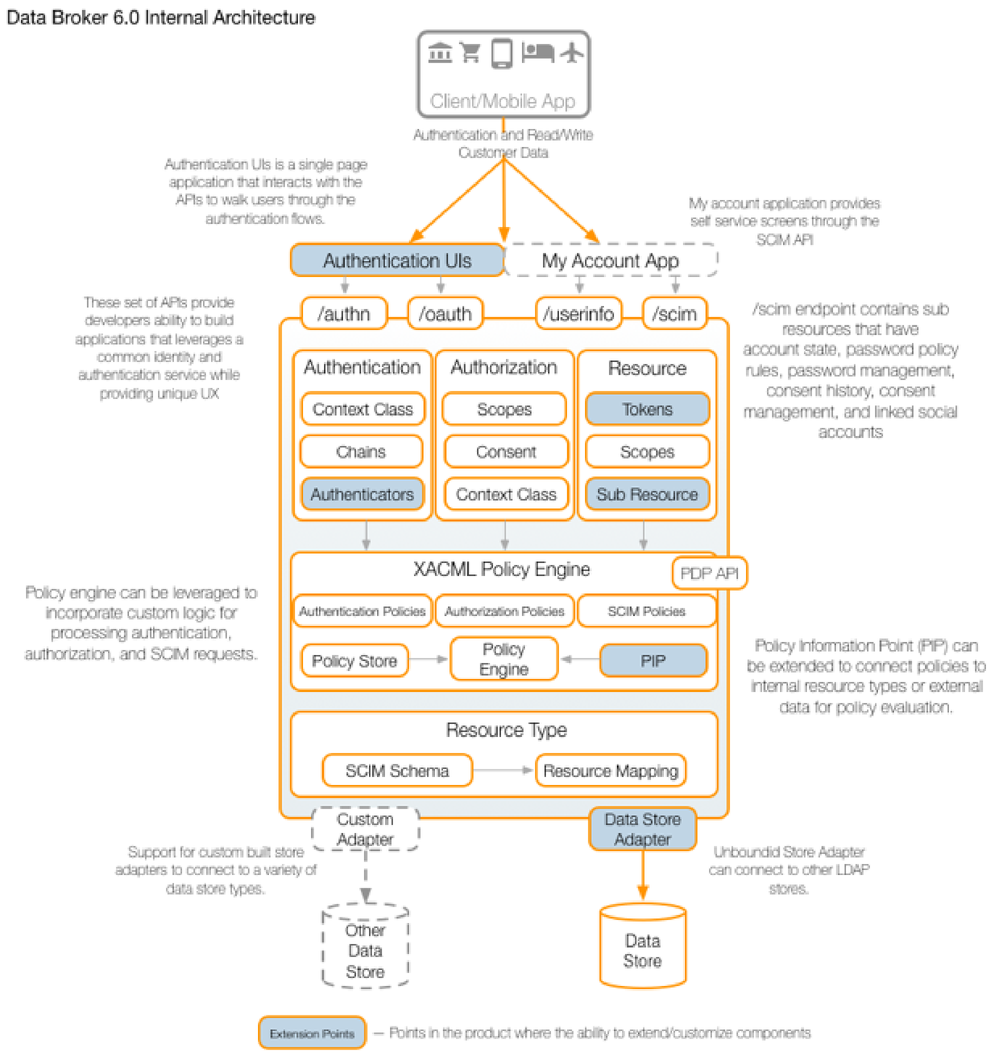
Data Governance Broker Architecture

The Data Governance 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 Governance Broker performs the following functions:

- Authorize an OAuth2 access token request, where the scopes requested represent resources that are served by the Data Governance 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 Governance 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 Governance Broker architecture and its components.


Data Broker 6.0 Internal Architecture
Client/Mobile App
Authentication and Read/Write Customer Data
Authentication UIs is a single page application that interacts with the APIs to walk users through the authentication flows.
My account application provides self service screens through the SCIM API
Authentication UIs
My Account App
/authn
/oauth
/userinfo
/scim
These set of APIs provide developers ability to build applications that leverages a common identity and authentication service while providing unique UX
/scim endpoint contains sub resources that have account state, password policy rules, password management, consent history, consent management, and linked social accounts
Authentication
Context Class
Chains
Authenticators
Authorization
Scopes
Consent
Context Class
Resource
Tokens
Scopes
Sub Resource
XACML Policy Engine
PDP API
Authentication Policies
Authorization Policies
SCIM Policies
Policy Store
Policy Engine
PIP
Policy engine can be leveraged to incorporate custom logic for processing authentication, authorization, and SCIM requests.
Policy Information Point (PIP) can be extended to connect policies to internal resource types or external data for policy evaluation.
Resource Type
SCIM Schema
Resource Mapping
Custom Adapter
Data Store Adapter
Support for custom built store adapters to connect to a variety of data store types.
Unboundid Store Adapter can connect to other LDAP stores.
Other Data Store
Data Store
Extension Points
— Points in the product where the ability to extend/customize components

Planning a Data Governance Broker deployment should start with determining its role as a Resource server. This includes defining what data can be accessed and updated from backend Directory Server, 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 Directory Servers 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 Governance 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 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 Governance 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 Governance Broker Configuration Overview

Data Governance Broker configuration defines all server services, policies, applications, resources, and the mapping of data from one or more backend Directory Servers. Configuration can be done from the command line with the `dsconfig` tool or through the Administrative 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 Administrative 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 Governance Broker needs to process requests as an identity provider or through an external identity provider. If multiple Data Governance 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:

- **Access Token Providers** – Defines how identity and authorization information is delivered by OAuth2 access tokens. The JWT Access Token Provider is provided as a default, if the `create-initial-broker-config` tool was used after Data Governance Broker installation. The access token provider is used by the `access-token-provider` property of the OpenID Connect Service.
- **Account Flow Handlers** – Defines account flow requirements prior to the authorization of requested data. Available flows include Password Recovery, Username Recovery, and Verify Account.
- **Authentication Chains** – Defines an authentication process in which a user must supply credentials for one or more Identity Authenticators as listed in the chain. Available chains include Account Verification, Login, Second Factor, and Username and Password Recovery.
- **Authentication Context Classes** – Defines a set of authentication requirements that must be met before access to OAuth2 scopes can be granted. A default class and multi-factor authentication (MFA) class are available.
- **Authentication Service** – Defines the settings for user authentication and session management.
- **External Identity Providers** – Specifies the identity providers that can be used to log into the Data Governance Broker, such as Google, Facebook, an OIDC provider, or a SAML provider.
- **Identity Authenticators** – Defines the authentication schemes that can be used to log into the Data Governance Broker. This is required by the Identity Provider Service.
- **OAuth2 Clients** – 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.
- **OpenID Connect Service** – Defines the OpenID Connect properties that the Data Governance Broker will use including access token issuer and duration, and the Authentication Context Classes (ACRs) that will be used.
- **Telephony Messaging Providers** – Defines the method for delivering messages about a user's account by telephone, such as SMS or voice.
- **Verification Code Generators** – Defines the process for generating a verification code for account details, such as email address or phone number. These can be used by Identity Authenticators.

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 Governance Broker. Any changes to these settings are saved to all Data Governance Brokers in a topology.

- **SCIM Resource Types** – Defines attribute mapping from a SCIM schema to native attributes found in Directory Server entries, which provides a unified view of identity data found in multiple Directory Servers. 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 Directory Server. 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 Directory Servers. Schemas provide the basis for creating SCIM Resource Types.
- **SCIM Sub Resource Type Handlers** – Defines a SCIM sub resource type that can be used to process extended operations on a SCIM resource type, such as validating an email address or making sure an account password meets specified requirements.

Data Sources

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

- **External Servers** – Lists the LDAP Directory Server instances that are configured with the Data Governance 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. This is an advanced setting.
- **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. This is an advanced setting.
- **Store Adapters** – Provides a Directory Server interface to the Data Governance Broker. Changes or additions to Store Adapters are saved to all Data Governance Brokers in a topology. Third-party store adapters can be created with the Server SDK.

Authorization and Policies

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

- **Access Token Validators** – Validates an access token used to access protected resources (OAuth2 scopes). Validators are used to decode tokens and return token metadata. The Data Governance Broker's local access token validator can be used, or a third-party token validator can be defined using the Server SDK.
- **OAuth Scopes** – Specifies the data being requested with an OAuth2 authorization request from an OAuth2 client.
- **Policy Information Providers** – Retrieves XACML attributes from a Policy Information Point (PIP) for policy evaluation. This is an advanced setting.
- **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 Governance 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 Governance 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. Most are not mirrored across a topology, unless otherwise stated. See General Server Configuration for more information.

- **Connection Handlers** – Defines the settings for handling all interaction with the clients, including accepting connections, reading requests, and sending responses.
- **Global Configuration** – Specifies the SMTP server, password policies, and LDAP request criteria configured for this server.
- **Key Manager Providers** – Manages the key material used to authenticate to another server. This is an advanced setting.
- **Key Pairs** – Defines the key pair that can be used to provide credentials for digital signatures. An existing key pair can be imported or a new one can be generated by the server. This configuration object is mirrored across a topology.
- **Locations** – Lists the locations in which servers that are accessed by the Data Governance Broker reside.
- **Trust Manager Providers** – Determine whether to trust certificates presented to the server. This is an advanced setting.

- **Trusted Certificates** – Specifies a trusted public key that can be used to verify credentials for digital signatures and public-key encryption. This configuration object is mirrored across a topology.

Web Services and Applications

These settings define the HTTP connection criteria for application access to the Data Governance 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.

- **HTTP Configuration** – Defines configuration for the Data Governance Broker HTTP Service. This is an advance setting and cannot be changed other than to include stack traces in error pages.
- **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.
- **Web Application Extensions** – Specifies the configuration settings for the Administrative Console and any other web applications that are configured to work with the Data Governance Broker.

LDAP Administration and Monitoring

These are all advanced settings to manage the local server's accounts, account requirements and security settings, and backend configuration. 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.

Logging, Monitoring, and Notifications

These settings define the notification criteria for system alerts, and the logging criteria for actions within the Data Governance 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.

- **Alarm Manager** – Defines the severity of alarms to be raised.
- **Alert Handlers** – Specifies the Alert Handlers used to notify administrators of problems or events that occur in the Data Governance Broker.
- **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.
- **LDAP SDK Debug Logger** – Records debug messages generated by the LDAP SDK for Java. This is an advanced setting.
- **Log File Rotation Listeners** – Defines an action for the server to take before a log file is rotated out of service, such as copying the file to a new location. This is an advanced setting.
- **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.
- **Monitor Providers** – Provides information about the state of the server or server components.

Sample Data Governance Broker Configuration

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

Data Governance Broker as both a Resource and Identity Provider Server

The following is a sample workflow for the Data Governance 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 Directory Server, or third-party adapters can be created with the Server SDK. A SCIM Schema can be used to surface attributes in the Directory Server, and is needed if mapping attributes from multiple Directory Servers 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 Governance Broker. Settings enable the OpenID Connect and OAuth2 functionality, login and second factor authentication flows, 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 Governance 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 Governance 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 Directory Server 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 Governance Broker configuration, a custom Policy Information Provider can be configured with the SDK, or with the help of Ping Identity Professional Services.
7. Policies determine what and how OAuth2 clients access resources. Make sure that policy rules work as expected by using Log Publishers to verify that the requests to and responses from the Data Governance 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, SAML, or OpenID Connect) for end users to log into the Data Governance Broker.
10. If there is an PingData Data Metrics Server installed, it can be configured to display system and consent metrics for the Data Governance Broker. See the *PingData Data Metrics Server Administration Guide* for information about configuring the Data Metrics Server.

Data Governance Broker as a Resource Server Only

If using the Data Governance 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 Governance 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 Governance Broker are configured to use the OAuth2 scopes that are installed as well. Both can be customized.

4. Since the Data Governance Broker is not the Identity Service Provider, a custom Policy Information Provider must be configured with the 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 Log Publishers to verify that the requests to and responses from the Data Governance 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, SAML, or OpenID Connect) accounts to access the Data Governance Broker.

Chapter 2: Installation

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

Topics include:

Installation Prerequisites

The following are required before installing the Data Governance Broker:

- Java 7
- Minimum of 2 GB RAM
- PingData Directory Server 5.2

Supported Platforms

The Data Governance 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 Ping Identity 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_DESCRIPTORS` environment variable, or by creating a `config/num-file-descriptors` file with a single line such as, `NUM_FILE_ DESCRIPTORS=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.

Note

For RedHat 7 or later, modify `the 20-nproc.conf` file to set both the open files and max user processes limits:

```
/etc/security/limits.d/20-nproc.conf

Add or edit the following lines if they do not already exist:
*     soft     nproc        65536
*     soft     nofile       65536
*     hard     nproc        65536
*     hard     nofile       65536
root  soft     nproc        unlimited
```

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 Governance 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`.

Disable Filesystem Swapping

Any performance tuning services, like `tuned`, should be be disabled. If performance tuning is required, `vm.swappiness` can be set by cloning the existing performance profile then adding `vm.swappiness = 0` to the new profile's `tuned.conf` file in `/usr/lib/tuned/profile-name/tuned.conf`. The updated profile is then selected by running `tuned-adm profile customized_profile`.

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.

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

Enabling the Server to Listen on Privileged Ports on Linux

Linux systems do not provide a direct analog to the Solaris User and Process Rights Management subsystems. Linux does have a similar mechanism called capabilities used to grant specific commands the ability to do things that are normally only allowed for a root account. This is different from the Solaris model because instead of granting the ability to a specific user, capabilities are granted to a specific command. It may be convenient to enable the server to listen on privileged ports while running as a non-root user.

The `setcap` command is used to assign capabilities to an application. The `cap_net_bind_service` capability enables a service to bind a socket to privileged ports (port numbers less than 1024). If Java is installed in `/ds/java` (and the Java command to run the server is `/ds/java/bin/java`), the Java binary can be granted the `cap_net_bind_service` capability with the following command:

```
$ sudo setcap cap_net_bind_service=+eip /ds/java/bin/java
```

The java binary needs an additional shared library (`libjli.so`) as part of the Java installation. More strict limitations are imposed on where the operating system will look for shared libraries to load for commands that have capabilities assigned. So it is also necessary to tell the operating system where to look for this library. This can be done by creating the file `/etc/ld.so.conf.d/libjli.conf` with the path to the directory that contains the `libjli.so` file. For example, if the Java installation is in `/ds/java`, the contents of that file should be:

```
/ds/java/lib/amd64/jli
```

Run the following command for the change to take effect:

```
$ sudo ldconfig -v
```

Installing the JDK

The Data Governance Broker requires the Java 64-bit JDK. Even if Java is already installed, create a separate Java installation for use by Data Governance Broker to ensure that updates to the system-wide Java installation do not inadvertently impact the Data Governance 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 Governance Broker generated tokens and User Store metadata. All Data Governance 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 Governance 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 Governance Broker installation, at least one PingData Directory Server 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 Governance 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 Directory Servers.

Any LDAP Directory Server added after the initial Data Governance Broker installation must be configured with the `prepare-external-store` tool before it can be used as a user store. See Data Governance Broker Installation Tools.

Installing the Directory Server

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

Note

All sensitive data in the user store can be encrypted. When using the Ping Identity Directory

Server as the user store, server-level encryption can be enabled as described in the "Encrypting Sensitive Data" section in the *PingData Directory Server 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 Directory Server:

1. Download the Directory Server zip distribution, `PingDirectory-<version>.zip`.
2. Unzip the file in any location.

   ```
   $ unzip PingDirectory-<version>.zip
   ```

3. Change to the top level PingDirectory folder.

   ```
   $ cd PingDirectory
   ```

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 Directory Server, 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 Directory Server repository, for example `dc=company,dc=com`.
16. Select an option to populate the database.
17. If this machine is dedicated to the Directory Server, tune the JVM memory allocation to use the maximum amount of memory the **Aggressive** option). This ensures that communication with the Directory Server 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 Governance Broker Installation Tools

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

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

Installing the Data Governance Broker

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

- An administrative account for the Data Governance Broker.
- An available port for the Data Governance Broker to accept HTTPS connections from REST API clients.
- An available port for the Administrative Console'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 Governance Broker:

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

   ```
   $ unzip PingDataGovernance-<version>.zip
   ```

3. Change to the top level PingDataGovernance 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 Governance Broker topology. For an initial installation, press **Enter** to accept the default (no). When adding additional Data Governance 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 Governance Broker, or press **Enter** to accept the default (local hostname).
8. Create the initial root user DN for the Data Governance 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 Governance Broker REST APIs and the Administrative Console to accept HTTPS connections, or press **Enter** to accept the default.
11. Enter the port to accept LDAP client connections, or press **Enter** to accept the default.
12. To enable LDAPS connections press **Enter** and enter a port, or type **no**.
13. To enable StartTLS connections over regular LDAP connection press **Enter**, or type **no**.
14. 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.
15. If this machine is dedicated to the Data Governance 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.
16. Enter a location name for this server. The location is used for failover purposes if this server belongs to a server group.

17. Enter an instance name for this Data Governance 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.
18. Press **Enter** (yes) to start the server when the configuration is applied.
19. 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 Governance 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 Governance Broker. Having the following in place will expedite the configuration:

- At least one Ping Identity Directory Server is installed. Have the host name and communication port available.
- Any additional Directory Servers that act as user stores. Only PingData Directory Servers 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 Governance 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 Governance 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 Governance 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 Governance 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 Directory Server. The connection is verified.
6. Select the location name for the Directory Server (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

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 Directory Server. This is the root account created in the initial setup, such as the default (cn=Directory Manager. The Data Governance Broker sets up the DN and tests that it can access the account. This is also the account used to log into the Administrative 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 Directory Server 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. The instructions for configuration in this guide use the standard user 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 Governance Broker and written to the broker-cfg.dsconfig file.

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

Logging into the Administrative Console

After the Data Governance Broker is installed, access the Administrative Console, https://<host>/console/login, to verify the configuration and manage the server. The root user DN or the common name of a root user DN is required to log into the Administrative Console. For example, if the DN created in Configuring the Data Governance Broker was cn=Directory Manager, directory manager can be used to log into the Administrative Console.

If the Administrative Console needs to run in an external container, such as Tomcat, a separate package can be installed according to that container's documentation. Contact Ping Identity Customer Support for the package location and instructions.

Installing Additional Data Governance Brokers in a Topology

A Data Governance 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 Governance Broker's local configuration and links the two configurations. The configuration of Data Governance 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 Governance 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 Governance 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 Governance Broker topology.
5. Enter the host name of the peer Data Governance Broker server from which the configuration will be copied.
6. Enter the port of the peer Data Governance Broker.
7. Choose the security communication to use to connect to the peer Data Governance Broker.
8. Enter the manager account DN and password for the peer Data Governance 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 Governance Broker, or press **Enter** to accept the default.
11. Select the option to install the Administrative Console application, if desired.
12. Enter the HTTPS connection port for the Administrative Console application, or press **Enter** to accept the default.
13. Enter the port on which the new Data Governance Broker will accept connections from LDAP clients, or press **Enter** to accept the default.
14. Choose a certificate option for this Data Governance 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.
update.bat, and update	The update tool for UNIX/Linux systems and Windows systems. (Update is not supported for version 6.0)
uninstall.bat, and uninstall	The uninstall tool for UNIX/Linux systems and Windows systems.
vendor_logo.png	The image file for the Ping Identity 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-notices	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 PingData Data Metrics Server.
bin	Stores UNIX/Linux-based command-line tools.
bat	Stores Windows-based command-line tools.
webapps	Stores the Administrative Console .war file, the Authentication interface reference application's war file and source, 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 Data Metrics Server.
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 ldif files, make-ldif 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 Governance Broker uses the `setup` and `create-initial-broker-config` 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 PingData Directory Server, the `prepare-external-store` tool is run. This updates the Directory Server’s schema, creates a privileged service account for use by the Data Governance 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 server’s schema, and creates a privileged service account for use by the Data Governance 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.

Note

The Directory Server ACIs may need to be configured to grant access to elements of data, or specific LDAP controls using ACIs, depending on which Data Governance Broker services are used. See `resource/starter-schemas/README.txt` for sample ACIs.

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 Directory Servers. See Installing the Directory Server.
2. Run the Data Governance Broker `setup` command on the server that will host the Data Governance Broker.

   ```
   $ ./setup --cli --no-prompt --acceptLicense --maxHeapSize 2g \
     --ldapPort 2389 --ldapsPort 2636 --httpsPort 8443 \
     --location Austin --instanceName broker1 \
     --rootUserPassword <password> \
     --useJavaTrustStore <path>/keystores/truststore.jks \
     --useJavaKeystore <path>/keystores/broker1keystore.jks \
     --trustStorePasswordFile<path>/keystores/truststore.txt \
     --keystorePasswordFile <path>/keystores/keystore.txt \
     --certNickname server-cert
   ```

   The `--trustStorePasswordFile` option is only required if this server is expected to update the truststore with certificates of other servers in the topology.

   The password for the private key associated with the certificate (`server-cert`) should be the same as the keystore password.
3. Run `prepare-external-store` for each user store.

   ```
   $ ./prepare-external-store --no-prompt \
     --hostname ds1.example.com \
     --port 1636 --useSSL --trustStorePath <path>/keystores/truststore.jks
   \
     --userStoreBaseDN "ou=people,dc=example,dc=com" \
     --brokerBindPassword <password> \
     --bindDN "cn=directory manager" \
     --bindPassword <password>
   ```
4. Run the `create-initial-broker-config` tool.

   ```
   $ ./create-initial-broker-config --no-prompt \
     --port 2636 --useSSL --trustStorePath <path>/keystores/truststore.jks
   \
     --bindDN "cn=Directory Manager" \
     --bindPassword <password> \
     --brokerBindPassword <password> \
     --externalServerConnectionSecurity useSSL \
     --userStoreBaseDN "o=people,dc=example,dc=com" \
     --userStore ds1.example.com:1636:Austin
   ```

Installing Sample Users

The Data Governance Broker provides a template to create a set of users (1000) that can be used by the sample application. The schema must be created from `<server-root>/resource/starter-schemas/reference-apps-make-ldif.template` and installed on the Ping Identity Directory Server. 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 Directory Server entries according to the directives in the LDIF file:

1. From the Directory Server server root, stop the server.

   ```
   $ <PingDirectory>/bin/stop-ds
   ```

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

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

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

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

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

   ```
     --overwriteExistingEntries
   ```

Run the Data Governance Broker

To start the Data Governance 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 Governance Broker in the foreground:

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

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

2. Stop the Data Governance 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 Governance Broker

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

Schedule a Server Shutdown

The Data Governance 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 Governance 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 Governance Broker

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

Note

If a Data Governance 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 Governance 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 Governance Broker Sample Application

A sample application can be installed in addition to the Administrative Console application to determine if account flow activities are configured correctly. This application is designed to illustrate a client's view of how data can be requested and delivered from the Data Governance Broker. The sample application is located in `<server-root>/samples/my-account.tar.gz`. Extract the file and follow the instructions in the `README.md` file to install the sample application.

To use this application with the Data Governance Broker's starter schema, sample users must be installed in the configured PingData Directory Server. See Installing Sample Users for details.

Note

Because the sample applications use the Data Governance 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.

Chapter 3: Data Access and Mapping

Directory Servers provide the resources that can be accessed by OAuth2 clients. Attributes can be mapped from multiple Directory Servers 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

When a Directory Server is configured, a Store Adapter is installed to read and return native SCIM objects. Custom store adapters can be created for non-LDAP Directory Servers with the Ping Identity 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 Governance 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 Governance Broker with an Ping Identity Directory Server or other external directory. For example, an LDAP Store Adapter manages the attribute mappings from an LDAP Directory Server to a SCIM schema used for a corresponding SCIM Resource Type. The Data Governance 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 Directory Servers 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.

Directory Servers

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

When a Store Adapter is added to the Data Governance 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 Directory Server stores a user's email address, and an email address can always be considered a primary key (that is, it is always unique per use), 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 Governance 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 Governance Broker provides a default store adapter that supports LDAP directory servers. Custom store adapters can be written using the 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.

Note

The Directory Server ACIs may need to be configured to grant access to elements of data, or specific LDAP controls using ACIs, depending on which Data Governance Broker services are used. See `resource/starter-schemas/README.txt` for sample ACIs.

Creating custom store adapters requires the Server SDK. See Server Extensions for information.

Primary and Secondary Store Adapters

If the Data Governance Broker is used to aggregate user attributes from multiple Directory Servers, secondary store adapters can be configured. Store adapters contain the configuration that the Data Governance Broker uses to interact directly with external Directory Servers. Every Directory Server 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 Governance 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.

Note

When creating SCIM resources backed by secondary store adapters, the server automatically sets the secondary correlation attribute value if it does not already have a value from the resource create request.

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

```
$ 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 Directory Servers 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 ping-identity-ds \
  ...
```

2. Configure each server in Set B.

```
$ dsconfig create-external-server \
  --server-name demographics-server \
  --type ping-identity-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 Directory Server. 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 Governance 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 Governance 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:pingidentity:schemas:sample:profile:1.0:topicPreferences` is mapped to the LDAP attribute `ubidXTopicPreferenceJSON` from the Marketing Directory Server 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 Governance 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 Governance 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 Governance Broker returns an error. For each candidate search result from a store adapter, the Data Governance 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 Governance 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 Governance Broker functions rely on this:

- The Match Filter property of the default Username Password Identity Authenticator.
- The Store Adapter Mapping for the `userName` attribute of the default starter schema.

It may be the case that the attribute used for the username is also an RDN attribute in the Directory Server. 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 PingData Directory Server provides the attribute uniqueness plugin that can be used if configuration changes are required. See the *PingData Directory Server Administration Guide*.

Defining SCIM Resource Types

SCIM Resource Types provide a unified view of resources between the Data Governance Broker and one or more underlying Directory Servers, and correspond to the SCIM 2.0 SCIM Resource Type. SCIM Resource Types determine what resources can be accessed from a Directory Server. 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, Directory Server 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 PingData Directory Server.

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 Directory Servers. 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:

```
$ 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:pingidentity:schemas:User:1.0 \
  --set required-schema-extension:urn:pingidentity:schemas:sample:profile:1.0
```

SCIM Resource Types can also be configured in the Administrative 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:pingidentity: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 Administrative 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 Directory Server. Store Adapter Mappings define a mapping between SCIM Resource Type attributes and store adapter attributes. When the Data Governance Broker is installed with an PingData Directory Server, the schema attributes are automatically mapped to a User SCIM Schema Resource Type.

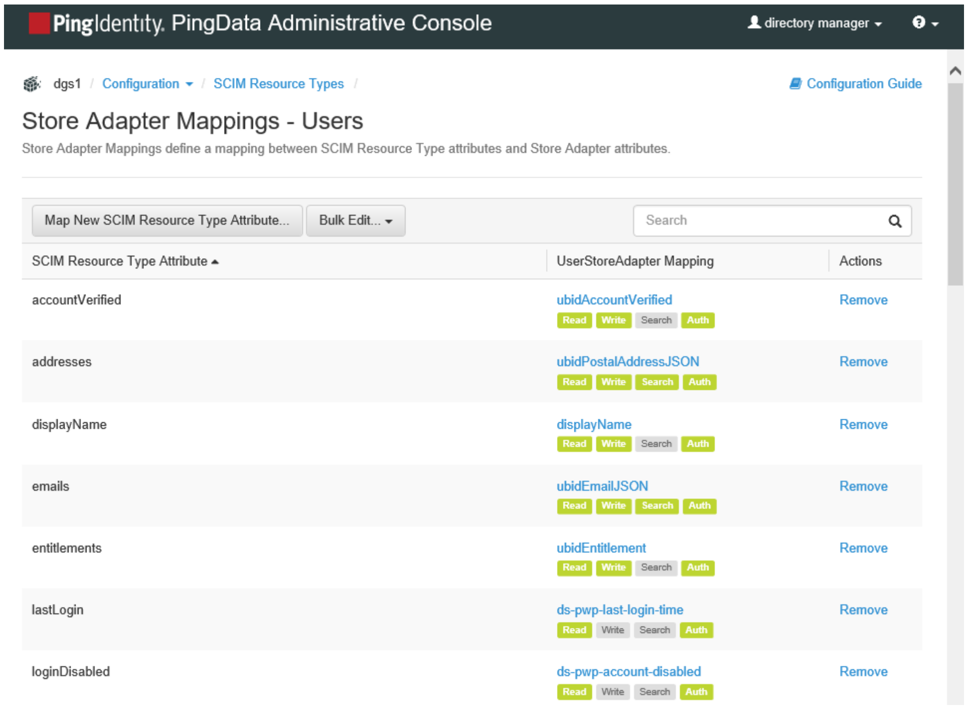
Note

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 Administrative Console through **SCIM -> SCIM Resource Types**. Click **Actions -> Edit Store Adapter Mappings** for a SCIM Resource Type. The following is displayed:


PingIdentity. PingData Administrative Console
directory manager
dgs1 / Configuration / SCIM Resource Types /
Configuration Guide
Map New SCIM Resource Type Attribute...
Bulk Edit...
Search

Store Adapter Mappings - Users

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

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
loginDisabled	ds-pwp-account-disabled 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 Directory Server. Indexed Directory Server 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 Directory Server, it should not be marked as Searchable here.
- **Authoritative** – If there are multiple mappings for this attribute (from multiple Directory Servers), 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. See Creating an OpenID Connect Claims Map for details.

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.

For searches involving sub-attributes of SCIM attributes that are mapped to LDAP JSON attributes in the PingData Directory Server, the sub-attribute field names in the search filter are treated case-sensitively because the Directory Server treats them this way. This is a departure from the SCIM 2.0 specification, where attribute names in search filters are case-insensitive.

For example, the SCIM attribute `name` has the sub-attribute `familyName`. The SCIM attribute `name` is mapped to the LDAP JSON attribute `scimName`. The search filter `NAME.FAMILYNAME eq "Zweig"` will not return a search result for an entry containing the specified value `Zweig` in the `familyName` sub-attribute. A search result for this entry is returned if the filter is specified instead as `NAME.familyName`. This is because the top-level attribute can be matched case-insensitively but the sub-attribute can only be matched case-sensitively.

Defining SCIM Sub Resource Type Handlers

SCIM Sub Resource Type Handlers may be used to define a SCIM sub resource type that may be used to process extended operations on a SCIM resource type. Sub resource types are used to retrieve session meta data that the Data Governance Broker stores with a user entry, such as consent information or account state. They are mainly used in the resource server role to exposes a per resource API for managing metadata that can not be managed using standard SCIM attributes. Like Resource Types, they are assigned to OAuth2 scopes.

The following SCIM Sub resource Type Handlers are available:

Available Sub Resource Type Handlers

Sub Resource Type Handler	Description
Account State	Retrieves account state information based on the `accountUsabilityNotices`, `accountUsabilityWarnings`, `accountUsabilityErrors`, and `accountActivationTime` settings in the PingData Directory Server.
Consent	Updates a user's consent for an application to access resources.
Consent History	Retrieves the consent history for a user.
Email Address Validator	Validates the email address for a user, based on the last validated value. The Email Delivered Code Identity Authenticator uses the same metadata to determine if an email address is validated before using it for authentication. A create request can validate an attribute path with an existing value, or a new value by sending a verification code to the email address. If the same verification code is provided in a subsequent replace request, the attribute value is recorded as validated in the metadata.
External Identity	Updates or retrieves external identity linked information for an account.
Password Quality Requirements	Retrieves the password requirements for this account based on the password quality requirements extended operation settings in the PingData Directory Server.
Password Update	Updates an account password based on the password quality requirements extended operation settings in the PingData Directory Server.
Phone Number Validator	Validates the phone number for a user, based on the last validated value. A create request can validate an attribute path with an existing value, or a new value by sending a verification code to the phone number. If the same verification code is provided in a subsequent replace request, the attribute value is recorded as validated in the metadata. The Telephony Messaging Providers are used to send verification codes.

Available Sub Resource Type Handlers

Sub Resource Type Handler	Description
Session	Retrieves and can be used to remove a user's current sessions. The session will return information about when it was last used, the IP address, and UA string. When removing a session, all access tokens for the user are also revoked.
TOTP Shared Secret	Determines if a time-based one-time password (TOTP) shared secret is currently registered for a user. It can also be used to generate a new shared secret, validate it using the TOTP device, and register it on the user's resource based on the `has-totp-shared-secret` password policy state operation settings in the PingData Directory Server.

Creating a SCIM Sub Resource Type

SCIM Resource Types can be used to set or retrieve session meta data saved with a user's entry.

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

```
$ bin/dsconfig create-scim-sub-resource-type-handler \
  --handler-name Preferences  \
  --set "description:account preferences"  \
  --set enabled:true  \
  --set endpoint:accountPreferences  \
  --set java-
class:com.unboundid.directory.broker.api.AccountPreferencesSCIMSubResourceTypeHandler
```

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

Creating a SCIM Sub Resource Type

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

- A name for this SCIM Sub Resource Type.
- An optional description for the SCIM Sub Resource Type.
- The SCIM Sub Resource Type's endpoint HTTP address, which will be relative to the `/scim/v2` base URL.
- The path to the SCIM Sub Resource Type attribute, if needed.
- If a verification code is needed, the code generator, validity duration, and messaging provider may be required.
- The Java Class for the SCIM Sub Resource Type extension handler, such as `com.unboundid.directory.broker.api.<new>SCIMSubResourceTypeHandler`.

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 Governance Broker, or this process can be done in the Administrative 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 \
     --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/
```

Accessing Data

The Data Governance 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 Governance 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 Governance Broker. If a request to the Data Governance Broker is delivering partial results, it may be due to policy settings. See How Policy Affects Access to Scopes.

The Data Governance 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 Governance Broker. See Authentication for details.

Note

/Bulk and /Groups are not supported.

See the Authentication Developer Portal section on SCIM APIs at https://developer.unboundid.com/.

Chapter 4: Identity Provider Services and User Authentication

The Data Governance Broker supports the OpenID Connect Standard 1.0, which enables an OAuth2 client to use the Data Governance Broker as its Identity Provider. OpenID Connect enables the client to offload its user authentication function to the Data Governance Broker, which will prompt the end user for any number of authentication flows 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.

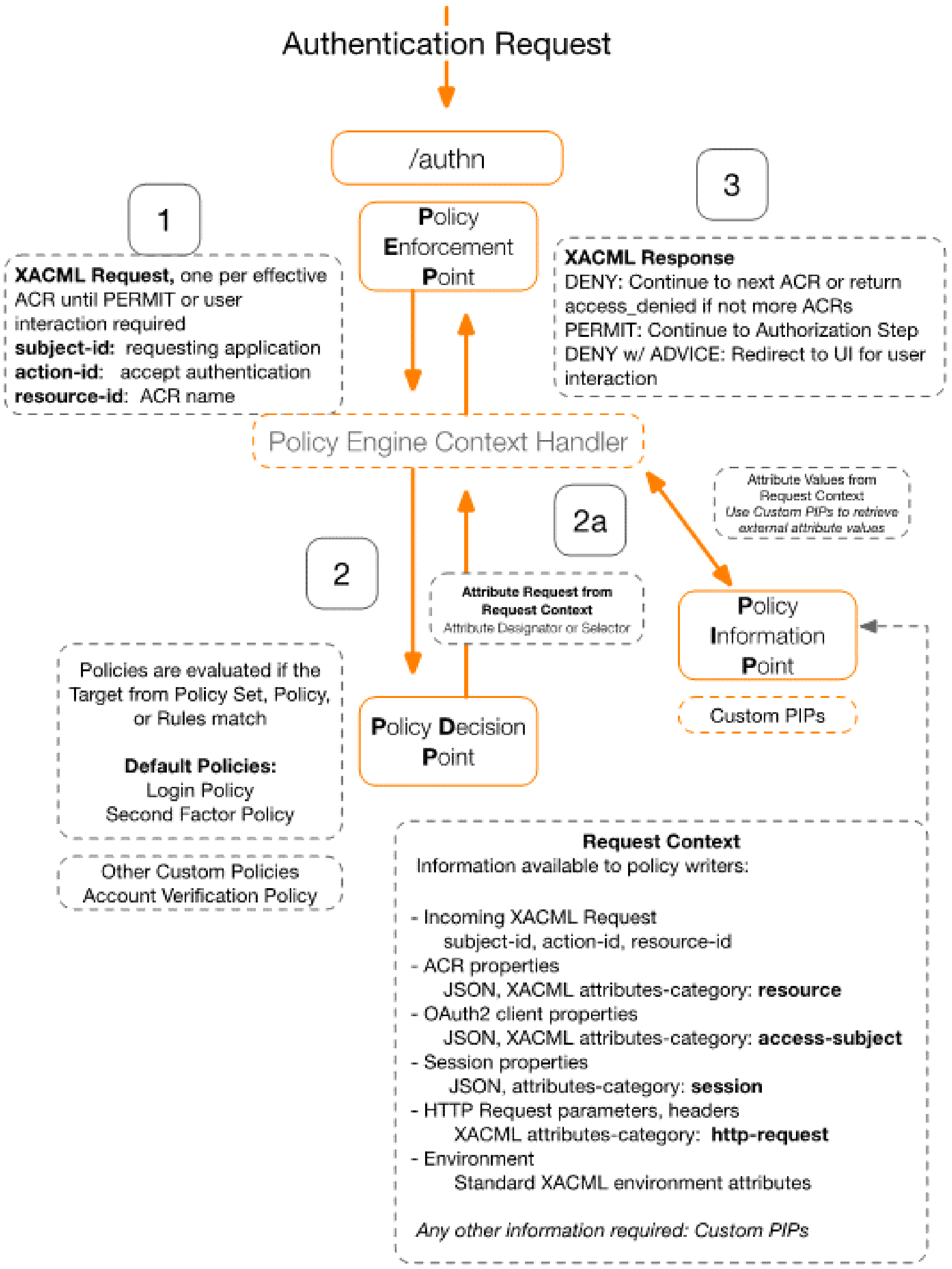
Topics include:

Authentication Processing Overview

When an OAuth2 client sends an OAuth2 or OpenID Connect authorization request to the Data Governance Broker’s authorize endpoint, the Data Governance Broker performs the following tasks:

- XACML policies are used to drive the authentication flow.
- XACML policies are evaluated again to authorize the scopes.
- Finally, an authorization code and/or access token is issued.

The following illustrates the authentication flow and components involved.

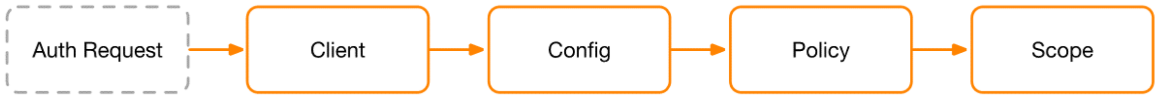

Authentication Request
/authn
1
Policy Enforcement Point
3
XACML Request, one per effective ACR until PERMIT or user interaction required
subject-id: requesting application
action-id: accept authentication
resource-id: ACR name
XACML Response
DENY: Continue to next ACR or return access_denied if not more ACRs
PERMIT: Continue to Authorization Step
DENY w/ ADVICE: Redirect to UI for user interaction
Policy Engine Context Handler
Attribute Values from Request Context
Use Custom PIPs to retrieve external attribute values
2a
2
Attribute Request from Request Context
Attribute Designator or Selector
Policy Information Point
Policies are evaluated if the Target from Policy Set, Policy, or Rules match
Default Policies:
Login Policy
Second Factor Policy
Custom PIPs
Policy Decision Point
Other Custom Policies
Account Verification Policy
Request Context
Information available to policy writers:
- Incoming XACML Request
subject-id, action-id, resource-id
- ACR properties
JSON, XACML attributes-category: resource
- OAuth2 client properties
JSON, XACML attributes-category: access-subject
- Session properties
JSON, attributes-category: session
- HTTP Request parameters, headers
XACML attributes-category: http-request
- Environment
Standard XACML environment attributes
Any other information required: Custom PIPs

Authentication Context

An Authentication Context represents the information known about how, when, and where the current user was authenticated. An Authentication Context Class reference (ACR) is a grouping of authentication requirements. For example, an ACR may specify that a particular authentication method must be used, or it may specify a maximum duration since the user was last authenticated by any method. At any point in time, it is possible to determine whether an ACR is satisfied, or not, by the current Authentication Context. An Authentication Context is evaluated by XACML policies to determine if it satisfies the requirements of an ACR.

The OAuth2 client can drive the Data Governance Broker's authentication behavior by specifying a set of ACRs that the Authentication Context must meet using the `acr` request parameter, as specified by the OpenID Connect specification. If the client did not specify the ACRs in the request, the configured OAuth2 client level or global level defaults are used to determine authentication requirements.

Data Broker's ACR Progression



The Data Governance Broker evaluates ACRs in the order specified. If an ID token has been requested by a client, once the policies return a permit for one of the ACRs, that ACR will be included in the `acr` claim of the ID token. Additionally, the configured authentication method references (AMRs) for all authenticators used to satisfy the ACR are included in the `amr` claim of the ID token as well. The following are properties of an ACR:

ACR Properties

Property	Description
Name	The name of the ACR, which cannot contain spaces.
Description	A description of the ACR for administrative use.
Login Authentication Chain	The Authentication Chain for user login.
Second Factor Authentication Chain	The Authentication Chain for second factor authentication.
Login Expiration Interval	The time at which the login will expire.
Second Factor Expiration Interval	The time at which the second factor authentication will expire.

Login and Second Factor Flows

The login and second factor authentication flows are designed to authenticate a user using an Authentication Chain specified by policy advice. If the policy doesn't specify the Authentication Chain to use, the configured global defaults are used. After successful invocation of the chain, the time of the last successful login/second factor, the chain used, client IP, and brower's UA string are recorded in a new session metadata in the user resource. A long lived cookie is set on the browser to associate it with this session metadata in future requests.

Authentication Chain

Each flow uses an Authentication Chain which is an ordered set of Identity Authenticators and their enforcement criteria. An Identity Authenticator is responsible for checking one or more credentials provided by the end-user in the request and returning a success or failure along with a response containing its current state. When a login or second factor request is processed, it invokes the Identity Authenticators in the order defined by the chain. Enforcement criteria is then applied based on the success/failure returned by each authenticator:

- `required-continue`: The authenticator is required to succeed. If it succeeds or fails, authentication still continues to proceed down the chain.
- `required-stop-on-failure`: The authenticator is required to succeed. If it succeeds, authentication continues down the chain. If it fails, authentication does not proceed down the chain and is considered failed.
- `optional-stop-on-success`: The authenticator is not required to succeed. If it does succeed, authentication does not proceed further and is considered successful. If it fails, authentication continues down the chain.
- `optional-continue`: The authenticator is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the chain.

Account Flow

Account flows provide the mechanism to perform pre-authentication account self-services such as account verification, consent agreement capturing, and other actions. The Data Governance Broker provides three account flow handlers:

Verify Account Flow Handler – Designed to use an authentication chain to validate a user's account. Upon success, it will allow the client to update the user resource to indicate the account is validated. This handler is used to validate a newly registered user's email address using the Email Delivered Code Identity Authenticator. The flow is triggered by a sample policy that checks if the user's `accountVerified` attribute is set to `TRUE`. If not, it will return `deny` with advice to perform this flow. Upon success, the user's `accountVerified` attribute is set to `TRUE`.

Username Recovery Flow Handler – Designed to use an authentication chain to locate and verify the user's identity before returning the value of the configured `username` attribute. By default, this handler is configured to use the reCaptcha, lookup, and Email Delivered Code Identity Authenticators to locate and verify the user to recover. This flow is only triggered by a link generated by the configuration of the Username Password Identity Authenticator used during login.

Password Recovery Flow Handler – Designed to use an authentication chain to locate and verify the user's identity before the user's password requirements are returned and a new password may be set. By default, this handler is configured to use the reCaptcha, lookup, and Email Delivered Code Identity Authenticators to locate and verify the user to recover. This flow is only triggered by a link generated by the configuration of the Username Password Identity Authenticator used during login.

Identity Authenticators

Identity Authenticators provide ways to authenticate a user or provide additional assurance about the identity of a user who is already authenticated.

Account Lookup Authenticator – Designed to be part of the authentication chain used by the username/password recovery account flows to locate the user to recover.

A configured SCIM filter template is used to perform a search operation to locate the server. Placeholders in the filter template will be replaced by the values of the authenticator’s request parameters.

The authenticator will return success if the search returned exactly one user resource.

Email Delivered Code Authenticator – Designed to be part of the authentication chain used by the login, second factor, as well as username/password recovery flows.

It uses the configured Verification Code Generator to generate a random code which will be sent to the user’s email address. The user’s email address is retrieved from the configured attribute path of the current user in the chain invocation context. The code is sent using the configured SMTP external server. The subject text of the message sent is specified in the authentication request.

The authenticator expects a follow up validation request with the generated code. The authenticator will then return success if the code matches.

Telephony Delivered Code Authenticator – Designed to be part of the authentication chain used by the login, second factor, as well as username/password recovery flows.

It uses the configured Verification Code Generator to generate a random code which will be sent to the user’s telephone number. The user’s telephone number is retrieved from the configured attribute path of the current user in the chain invocation context. The code is sent using the configured Telephony Messaging Provider. The text of the message sent is specified in the authentication request.

The authenticator expects a follow up validation request with the generated code. The authenticator will then return success if the code matches.

- Twilio SMS Messaging Provider: uses Twilio SMS API to send a SMS to the telephone number.
- Twilio Voice Messaging Provider: uses the Twilio Voice API to call the telephone number and read the message.

TOTP Authenticator – Designed to be part of the authentication chain used by the second factor flow. It uses the Validate TOTP Extended LDAP Operation to validate the TOTP in the request with an PingData Directory Server.

External Authenticator – Designed to be part of the authentication chain used by the login flow. This authenticator is used with the configured External Identity Providers, and returns the IDP’s authentication/authorization URL to the client (through the `auth-ui`). During the Identity Provider callback, the client is expected to relay it back to the authenticator in a follow up request.

If configured with a Directory Server, this authenticator checks and updates password policy state in the Directory Server using the Externally Processed Authentication SASL Mechanism. It supports returning `accountDisabled` and `accountLocked` errors based on the user’s password policy in the Directory Server.

Username Password Authenticator – Designed to be part of the authentication chain used by the login flow. The `username` parameter may be omitted. In this case, the password in the request will be checked against the current user in the chain invocation context.

Additionally, it now supports forced password change based on the account usability errors returned by the Directory Server (such as `must-change-password` or `password-expired-with-grace-logins`). In this case, the password requirements are returned in the response. It then expects a follow up request containing the current and new password. It then uses the Password Modify Extended LDAP operation to update the user's password. Lastly, it also supports returning `accountDisabled` and `accountLocked` errors based on the user's password policy in the Directory Server.

reCaptcha Authenticator – Designed to be part of the authentication chain used by the Login and Username Password Recovery flows. It should be configured to be invoked before other authenticators in the chain to prevent brute force attacks.

Registration Authenticator – Designed to be part of the authentication chain used by the Login flow. It creates a new user resource based on the attribute and value pairs provided in the request, and returns success. The initial state of this authenticator returns the password requirements for a new user.

Authentication Configuration Examples

The following are high-level steps to configure the following in the Data Governance Broker:

- Basic authentication with username and password.
- Adding an external identity provider for social login.
- Adding account registration.
- Adding second factor authentication.
- Adding account recovery.

Username Password Authentication

To configure basic authentication where the end user supplies a username and password, configure the following:

Identity Authenticators – After Data Governance Broker installation, a Username Password authenticator is available. Check the Match Filter setting to make sure that works with the schema in place.

Authentication Chains – If a login chain is not already created after installation, create a new one and call it Login. Select the Identity Authenticator (Username Password) configured previously. For Enforcement criteria, select `required-continue` if every authentication requires a username and password to be submitted. Enter an evaluation order index of `0` if this is the first authenticator add. The order index determines how to evaluate an authentication submission.

Authentication Service – Select the Login chain created previously. This is where the Data Governance Broker's login default is configured. If no other authentication requirements are configured or specified, this authentication chain is used.

Authentication Context Class – Create an ACR, which consists of authentication chains for Login and Second Factor authentication. Specify the intervals a user has to reauthenticate for each factor. An authentication chain does not have to be assigned an ACR. If the environment will have one possible authentication experience, creating an unpopulated ACR will be sufficient. The Data Governance Broker will rely on the login chains assigned under the Authentication Service.

OpenID Connect Service – Select which ACRs will be evaluated and in what order (top to bottom). An ACR must be selected for the Data Governance Broker to accept authentication requests, but it does not have to have authentication chains defined. Every authentication request has to be evaluated in the context of an ACR, so at least one has to be defined and selected.

OAuth2 Client – Add an Oauth2 client to the Data Governance Broker and add scopes for the client to access. Permitted ACRs can be used to specify the ACR evaluation the Data Governance Broker will make for authentication if no ACR is provided by the client. If an ACR is provided, this list acts as a filtering list to determine if the requested ACR is permitted by client. Permitted Scopes can be configured to determine if consent is required, if a scope is required to gain access to the client, and if a particular ACR requirement is needed to grant access to a scope.

Adding Social Login through an External Identity Provider

Have the necessary credentials and configuration information needed from the identity provider to configure the Data Governance Broker as a relying party. Depending on the provider, this may include client ID, client secret or certificates.

External Identity Providers – Configure connections to Google, Facebook, SAML, and OpenID Connect providers.

Identity Authenticators – Specify a name, URN, and any attributes that can be shared with the client for identification and possibly pre-populating a registration form. A single authenticator can be used by multiple identity providers.

Authentication Chain – Open the Login chain, add the new authenticator, set the evaluation order, and enforcement criteria such as `optional-stop-on-success`.

The `auth-ui` determines that an external identity provider is available as a login option and renders the appropriate icon to initiate that request. Icons are available for Facebook, Google, and OpenID Connect providers. A generic icon for is available for SAML.

Adding Account Registration

To add account registration, define authenticator, add it to an authentication chain, and modify or create an ACR:

Identity Authenticator – Create a Registration Identity Authenticator, and define which attributes need to be captured from a registration form. Review an External Identity Provider's mappings to make sure they align with the registration attributes required.

Authentication Chain – Add the Registration Identity Authenticator to the primary (login authentication) chain used in the context classes. There may be multiple authentication chains used in context classes as the login authentication chain.

Authentication UI – Modify the `auth-ui/app/register` source files to match the fields required for registration. They are built using PingData sample schema and will need to be customized. See Customizing the Authentication User Interface.

Adding Second Factor Authentication

To add an additional authentication requirement, add or configure an Identity Authenticator, add the authenticator to an Authentication Chain, and add or configure an Authentication Context Class. Available authenticators that can be used as second factors include:

- Email (Unverified/Verified)
- SMS (Unverified/Verified)
- TOTP (relies on a PingData Directory Server Validate TOTP Password Extended LDAP operation)

Identity Authenticator – Create a new Telephony Delivered Code Identity Authenticator, and define the schema URN, attribute path to find the phone number, the code generator, and whether validation is required.

Authentication Chain – Add this identity authenticator to a chain or create a new one for second factor only. For second factor, we recommend creating a separate chain so you can specify the second factors available to users.

Authentication Service – If a separate second factor chain was created, determine if second factor authentication is required by default. If the chain in the ACR does not specify a second factor chain, the Data Governance Broker will execute the defined chain for the Authentication Service.

Authorization Context Class – Add the second factor chain to the default context class and specify the interval in which a user will be prompted for that factor, or create a separate context class containing the second factor chain so that applications can specifically request second factor flows.

Authentication UI – If introducing a new identity authenticator, you will need to modify the authentication UI to recognize the new authenticator and present screens specific for that instance.

OAuth2 Client – Edit existing scopes to require a second factor ACR prior to granting the scope.

Adding Account Recovery

Account Flow Handlers are pre-authentication flows that interact with the user. These handlers are primarily used to process account recovery requests. There are two primary account recovery flows, username and password. To configure password recovery, configure the Password Recovery account flow handler:

Account Flow Handlers – Enable the account handler, and select or create an authentication chain that defines requirements necessary to determine if a user can change the account password, such as reCAPTCHA or Email Delivered Code. This account flow handler uses the Directory Server's change password extended LDAP operation if the authentication chain is successfully met.

Authentication Chain – Create a chain that defines the enforcement criteria to be considered before an action can be taken. Assign the chain to the Account Flow Handler.

OpenID Connect Request

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

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

client secret - A shared secret established between theData Governance 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 Governance Broker's HTTP endpoint addresses for authenticating the end user, obtaining authorization, and issuing and validating access tokens. See Data Governance 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. An `acr` parameter can be included to specify the authentication parameters the client requires.

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. The Data Governance 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 Governance 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 Governance Broker can only issue assertions about registered clients and user identifiers within its domain. The token is validated by the Data Governance 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 Governance Broker as a Relying Party

The Data Governance Broker, as relying party, acts as a client of an external identity provider service. Users can log into the Data Governance Broker with external identity provider accounts. The Data Governance Broker provides authentication claims, account linking, and profile retrieval services to the OAuth2 client. The Data Governance 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 Governance Broker's default login page for clients configured to use an external identity provider.

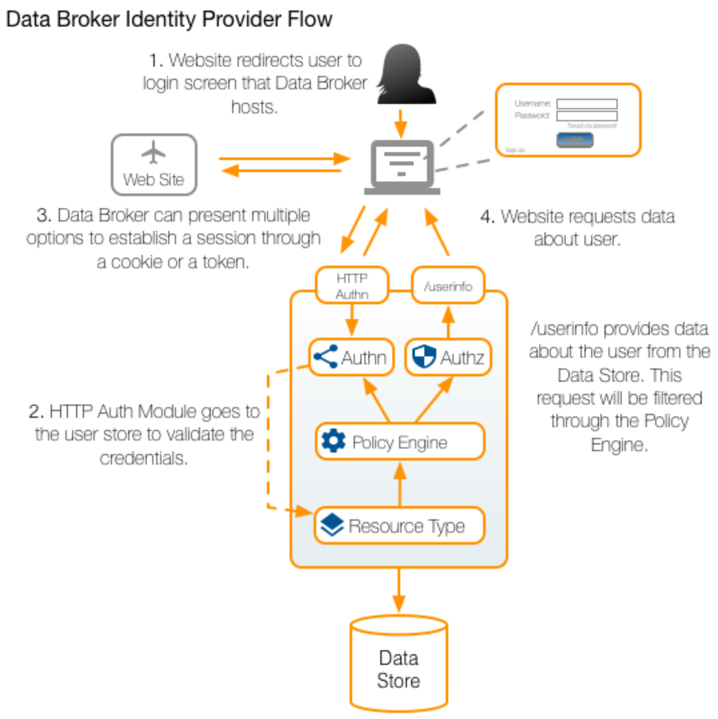
Creating an Account through Identity Provider Login

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

If a user attempts to log in with an external account, the External Identity Authenticator returns mapped attributes if the Data Governance Broker did not find a linked account for the user. The Authentication UI then presents the registration form so the user can register using the Registration Identity Authenticator.

Identity Provider Configuration

The Data Governance 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 Governance Broker, or the Data Governance Broker can rely on a configured external identity provider.


Data Broker Identity Provider Flow
1. Website redirects user to login screen that Data Broker hosts.
Web Site
3. Data Broker can present multiple options to establish a session through a cookie or a token.
4. Website requests data about user.
HTTP Authn
/userinfo
Authn
Authz
/userinfo provides data about the user from the Data Store. This request will be filtered through the Policy Engine.
2. HTTP Auth Module goes to the user store to validate the credentials.
Policy Engine
Resource Type
Data Store

The following components are configured:

- Access Token Providers – Defines how identity and authorization information is represented by OAuth2 access tokens. A default JSON Web Token Provider is available, or a new one can be created. To import or configure a new key pair, see Public and Private Key Store Configuration.
- Account Flow Handlers – Defines additional account flows after user authentication but before authorization to resources, such as username or password recovery.
- Authentication Chains – Defines an authentication process where the end-user must present credentials for one or more Identity Authenticators defined in the chain.
- Authentication Context Classes – Specifies a set of authentication context requirements (ACRs) that must be met before access to a resource (OAuth2 Scope) is granted.
- Authentication Service – Defines the properties that affect the Data Governance Broker OAuth2 service including access token settings and default ACRs.

- External Identity Provider – Configure Facebook, Google Plus, an OpenID Connect provider, or a SAML provider for authentication. Part of the External Identity Provider configuration is mapping attributes to SCIM Resource Types, and defining what should occur at login if there is a change to an attribute.
- Identity Authenticators – 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 Governance Broker.
- OAuth2 Clients – Define the OAuth2 clients that can request access to resources through the Data Governance Broker.
- OpenID Connect Claims – Defines a claim that can be exposed through the UserInfo endpoint, and its mapping to attribute(s) of the identity resource.
- OpenID Connect Service – Defines token requirements and the default set of ACRs that can be used to authenticate users.
- Telephony Messaging Providers – Delivers messages to users by telephone (SMS or voice message).
- Verification Code Generators – Deliver verification codes based on fixed-length strings built from one or more character sets.

Defining Access Token Providers

Access Token Providers define how identity and authorization information is represented by OAuth 2 access tokens. All tokens are signed JSON Web Tokens (JWTs) to enable verification from the Data Governance Broker or from an external resource server.

See Public and Private Key Store Configuration for key store settings. If not using the default provider, the desired key store configuration should be in place before configuring an Access Token Provider.

The following is a sample command line configuration:

```
$ bin/dsconfig create-access-token-provider \
  --provider-name "Signed JWT Access Token" \
  --type jwt --set "description:Default Token Provider" \
  --set "access-token-signing-key-pair:Access Token Signing Key Pair" \
  --set access-token-signing-algorithm:RS256
```

Access Token Providers can also be configured in the Administrative Console under **Identity Provider -> Access Token Providers**. The following information is needed to configure the Authentication Service:

- The unique name and optional description for this provider.
- The access token signing key pair.
- The access token signing algorithm, which can be either RS512, RS256, or RS384.

Defining Account Flow Handlers

Account Flow Handlers define additional account flows after authentication is complete, but before authorization to access resources. Three flow types are available:

Password Recovery – Used to reset a user's password.

Username Recovery – Used to look up a user's username.

Verify Account – Used to verify the contact information for a user using an authentication chain. The account is updated with a specified attribute to indicate successful verification.

This setting relies on the configuration of Authentication Chains.

The following is a sample command line configuration:

```
$ bin/dsconfig create-account-flow-handler \
  --handler-name "Username Recovery" \
  --type username-recovery \
  --set enabled:true \
  --set "authentication-chain:Username and Password Recovery" \
  --set username-attribute:username
```

Account Flow Handlers can also be configured in the Administrative Console under **Identity Provider -> Account Flow Handlers**. The following information is needed to configure an Account Flow Handler:

- A unique name and optional description for this flow.
- The Authentication Chain to use for this flow.
- For Username Recovery, the attribute to retrieve that is the username. For Verify Account, the attribute to set confirming that the account was verified.

Defining Authentication Chains

Authentication Chains define an authentication process where a user must present credentials for one or more Identity Authenticators defined in the chain. Identity Authenticators are listed in the order in which they should be performed. An authentication process will succeed or fail based on the enforcement criteria set for each Identity Authenticator.

The following is a sample command line configuration:

```
$ bin/dsconfig create-authentication-chain \
  --chain-name MFA
```

```
$ bin/dsconfig create-chained-identity-authenticator \
  --chain-name MFA \
  --authenticator-name "Verified Email Delivered Code" \
  --set evaluation-order-index:1
```

Authentication Chains can also be configured in the Administrative Console under **Identity Provider -> Authentication Chains**. The following information is needed to configure an Authentication Chain:

- The name and optional description for the chain.
- The set of Identity Authenticators to use in this chain.
- The enforcement criteria for the Identity Authenticators in this chain. Options are:
  - `required-continue` – The authenticator is required to succeed. If it succeeds or fails, authentication still continues to proceed down the chain.
  - `required-stop-on-failure` – The authenticator is required to succeed. If it succeeds, authentication continues down the chain. If it fails, authentication does not proceed down the chain and is considered failed.
  - `optional-stop-on-success` – The authenticator is not required to succeed. If it does succeed, authentication does not proceed down the chain and is considered success. If it fails, authentication continues down the Chain.
  - `optional-continue` – The authenticator is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the chain.
- The Evaluation Order Index. Chained Identity Authenticators are evaluated based on this index, from least to greatest.

Defining Authentication Context Classes

An Authentication Context Class (ACR) specifies the set of requirements that must be met before access to a resource (OAuth2 Scope) can be granted. With SAML and OpenID Connect providers, ACRs are defined around the property of assurance levels inferred from the methods used to authenticate the user.

The Authentication Context is evaluated by XACML policies to determine if it satisfies the requirements of an ACR. If the Authentication Context satisfies the ACR, the policy returns permit. However, if the Authentication Context can not currently satisfy the ACR as is, it may return a deny with advices to perform authentication flows.

The following is a sample command line configuration:

```
$ bin/dsconfig create-authentication-context-class \
  --class-name "New Class" \
  --set "description:Authentication class requiring login with a 12-hour expiration." \
  --set "login-authentication-chain:Second Factor" \
  --set second-factor-authentication-chain:MFA \
  --set "login-expiration-interval:1 d" \
  --set "second-factor-expiration-interval:5 s"
```

Authentication Context Classes can also be configured in the Administrative Console under **Identity Provider -> Authentication Context Classes**. The following information is needed to configure Authentication Context Classes:

- The name and optional description of the class.
- The Authentication Chain for user login.

- The Authentication Chain for second factor authentication.
- The time at which the login will expire.
- The time at which the second factor authentication will expire.

Defining the Authentication Service

The Authentication Service defines the properties that affect the Data Governance Broker session and authentication functions. These settings are used to define the actions that the Data Governance Broker can perform as an Identity Provider.

The following is a sample command line configuration:

```
$ bin/dsconfig set-authentication-service-prop \
  --set "login-authentication-chain:Second Factor" \
  --set "second-factor-authentication-chain:Account Verification" \
  --set "cookie-max-age:500 w" \
  --set max-concurrent-sessions:10
```

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

- The login and second factor authentication chains that will be used. By default, both are disabled.
- The cookie domain, path, and maximum age for each session.
- User session criteria, including max concurrent sessions, and flow inactivity timeout.
- The SCIM Resource Type that will provide the credentials and attributes of users that can be authenticated by the Data Governance Broker. This resource type must be configured with a primary LDAP store adapter connected to an PingData Directory Server or an PingData Directory Proxy Server. The Data Governance 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.
- The session resource attribute, which is exposed to the client during a valid user session. The client can use this as a personalized greeting during log in or second factor flows.

Creating an Identity Authenticator

Identity Authenticators define how a user can authenticate with the Data Governance Broker. The following Identity Authenticators are provided:

- **Account Lookup** – Used to lookup an end-user account from one or more specified request parameter values.
- **Linked External Identity** – Used to authenticate an end-user with an External Identity Provider.

- **reCAPTCHA** – Used to verify a user's response to a Google reCAPTCHA challenge.
- **Registration** – Used create and authenticate a new account from data entered by the end-user.
- **TOTP** – Used to authenticate an end-user with a time-based one-time password based on RFC 6238 by using the Directory Server Validate TOTP Password Extended LDAP operation.
- **Unverified Email Delivered Code** – Used to deliver a verification code to an e-mail address stored in a specified attribute from a user's SCIM resource, and then verify the code entered by the user.
- **Unverified Phone Delivered Code** – Used to deliver a verification code to a telephone number (by SMS or voice message) stored in a specified attribute of a user's SCIM resource, and then verify the code entered by the user.
- **Username Password** – Used to authenticate an end-user with a username and password using an LDAP BIND operation.
- **Verified Email Delivered Code** – Used to deliver a verification code to an e-mail address stored in a specified attribute from a user's SCIM resource and then verify the code subsequently entered by the user.
- **Verified Phone Delivered Code** – Used to deliver a verification code to a telephone number stored in a specified attribute of a user's SCIM resource, and then verify the code entered by the user.

The following is a sample command for creating a Username Password authenticator:

```
$ 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 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.

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 Authentication 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 Administrative 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.

Defining the OpenID Connect Service

The OpenID Connect Service defines token requirements and the default set of ACRs that can be used to authenticate users.

The following is a sample command line configuration:

```
$ bin/dsconfig set-openid-connect-service-prop \
  --set "authorization-code-validity-duration:2 m" \
  --set "access-token-validity-duration:10 h" \
  --set "refresh-token-validity-duration:2 w 2 d" \
  --set "id-token-validity-duration:10 m"
```

The OpenID Connect Service can also be configured in the Administrative Console under **Identity Provider -> OpenID Connect Service**. The following information is needed to configure the Authentication Service:

- 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 Governance Broker installation.
- The ordered list of Authentication Context Classes (ACRs), which determine the steps required for user authentication and access to scopes. A client can specify its own required ACRs, but they must be configured in the Data Governance Broker.
- The access token provider.

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 Governance Broker provides templates for creating Facebook and Google identity providers, or an OpenID Connect provider can be configured. All can be created through the Administrative 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 Governance 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.

Configuring a Redirect URI

An External Identity Provider's configuration may require a redirect-uri, such as `https://broker.example.com/auth-ui/callback.html`. The redirect URI value configured at the External Identity Provider will depend on the client application implementation. Login and registration through an External Identity Provider uses the `auth-ui` by default. If implementing a custom authentication UI, the redirect URI will reflect how the application is implemented.

The same is true for External Identity Provider linking through the SCIM API. For example If using the My-Account sample application, the redirect URI will resemble `https://broker.example.com/samples/my-account/callback.html`. However, if implementing a custom Data Governance Broker SCIM API client, the redirect URI will reflect the application's implementation.

Properties For Facebook

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

Properties For Google

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

Properties For OpenID Connect

- The Client ID that was given to the Data Governance Broker when it was registered with the identity provider.
- The Client Secret that was given to the Data Governance Broker when it was registered with the identity provider.
- The OpenID Connect scopes that can be requested from a registered Data Governance 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.

Properties For SAML

- The URL of the SAML Identity Provider Single Sign-On Service.
- The local entity ID used to populate the issuer field of outgoing SAML messages, which is the Data Governance Broker by default.
- The entity ID of the SAML Identity Provider, which must match the issuer field of the SAML messages received and the source ID of SAML Artifacts received.

Defining Telephony Messaging Providers

Telephony Messaging Providers are used to deliver messages to users by telephone (SMS or voice message). Two Twilio providers are included with the Data Governance Broker installation. A third-party provider can be created with the Server SDK.

The following is a sample command line configuration:

```
$ bin/dsconfig create-telephony-messaging-provider \
  --provider-name "New Provider" \
  --type sms-twilio \
  --set twilio-account-sid:<ID>
  --set twilio-auth-token:AAAd5IBOshLkg15Wqx7e6bNtWVndvs9DNXg=
  --set sender-phone-number:<0000000000>
```

Authentication Chains can also be configured in the Administrative Console under **Identity Provider -> Telephony Messaging Providers**. The following information is needed to configure a Telephony Messaging Provider:

- The name and optional description of the provider.
- The unique ID assigned to the Twilio account.
- The authentication token assigned to the account.
- The phone numbers obtained for use with the Twilio account.

Defining Verification Code Generators

Verification Code Generators are used to deliver verification codes based on fixed-length strings built from one or more character sets.

The following is a sample command line configuration:

```
$ bin/dsconfig create-verification-code-generator \
  --generator-name "New Code Generator" \
  --type random \
  --set code-character-set:alpha:ABCDEFG \
  --set code-character-set:numeric:0123456789 \
  --set code-format:"alpha:3,numeric:4"
```

Verification Code Generators can also be configured in the Administrative Console under **Identity Provider -> Verification Code Generators**. The following information is needed to configure a Verification Code Generator:

- The name and optional description of the generator.
- One or more named ASCII character sets used to generate codes.
- The authentication token assigned to the account.
- The format requirements of the generated code.

Chapter 5: 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 Governance 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 Governance Broker can redirect an end user back to the client when authorizing access to resources.

Topics include:

OAuth2 Client Considerations

Consider the following when configuring an OAuth2 client to connect with the Data Governance 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 are digitally signed JSON Web Tokens (JWT). In cases where external resource servers must validate access tokens, RSA public and private key configuration can be used to verify and trust Data Governance Broker tokens.
- 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. The token itself contains all the required information about the user, avoiding the need to query the database more than once.
- 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.
- The Data Governance Broker's Authentication API uses a cookie to track user sessions. Cookie management and server domains should be considered when deploying any clients that will use the Authentication API.

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 Governance 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 Governance Broker) to grant or deny access to a resource. If access is granted, the Data Governance 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 Governance Broker server. The Data Governance 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 Governance 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.

OAuth2 Authorization Response Types

The Data Governance 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.

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 Governance Broker will be needed by the OAuth2 client to request resources. Clients can be added through the Administrative 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:

General Information

- The name of the client.
- Optional description, and contact email address for this OAuth2 client.
- The contact email address for this client.
- The client URL, which must also be registered with the Data Governance Broker.
- The trusted origin(s) of the client if making JavaScript requests.
- The optional tag assigned to this client, for additional XACML policy processing.

OAuth2 Information

- The client ID and client secret can be generated by the Data Governance 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.
- If using the authorization code or implicit grant flow, specify a redirect URL.
- 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 Authentication Context Classes (ACRs) that are permitted for this client. If not specified, the client can determine the required ACRs in its request to the Data Governance Broker. By default, the Data Governance Broker provides two ACRs in the OpenID Connect Service Configuration. Any ACR that the client can require, must be defined in the Data Governance Broker.
- Any external identity providers that can be used to authenticate an end user account (advanced setting).
- The permitted scopes that can be requested by an OAuth2 client. These can also require additional ACRs, which are processed after the initial authentication phase of a client request. See Authentication Processing Overview for details.

The Data Governance 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=Splxl0BeZQQYbYS6WxSbIA&redirect_
uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

Response

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

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

{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "bearer",
    "expires_in": 3600,
    "scope": "openid email profile",
    "id_token": "eyJhbGciOiJIUzI1NiJ9.eyJhdXRoX3RpbWUiOjE0MjE4ODExMDMsImV4
                  cCI6MTQyMTg4MjAwOSwic3ViIjoiOWY4YTIzLWNjY2M3NmVlLWQwN2ItM2I
                  4Yy05MjJjLWRkZDgwOWM0YzE3MyIsImF1ZCI6WyJhY21lIl0sImlzcyI6Im
                  h0dHBzOlwvXC94MjI1MC0wMS5leGFtcGxlLmNvbSIsImlhdCI6MTQyMTg4M
                  TEwOX0.CZYpxocXZ-_DEPttmHqSiQ1FU8Pplb8I-7oK3PMp4-Y"
}
```

Token Validation by the Data Governance Broker

The Data Governance 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 Governance 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 Governance 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 Governance 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 Governance 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 Governance Broker

The token revocation endpoint (`/oauth/revoke`) enables OAuth2 clients to send a POST request to the Data Governance Broker to revoke access or refresh tokens. Revoking a token does not remove any associated consents. Token revocation conforms to the *OAuth 2.0 Token Revocation RFC 7009*.

During the revocation process, the Data Governance 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 Governance Broker revokes or invalidates the token.

For example, he following revokes a token:

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

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

Token Validation by an External Resource Server

Data Governance Broker access tokens are digitally signed JWTs, which enables them to be validated by the Data Governance Broker or by an external resource server with the configuration of public/private key pairs.

When a client application makes a request to the Data Governance Broker for access to resources, a JWT is returned and must be saved locally (in local storage, or through cookies), instead of the traditional approach of creating a session in the server and returning a cookie.

When a user wants to access a protected resource, the client application sends the JWT, typically in the Authorization header using the Bearer schema. The resource server (either the Data Governance Broker or an external server) checks for a valid JWT in the Authorization header. If it is present, the access to protected resources is granted. The authentication mechanism is stateless, as the user's state is never saved in server memory.

Because JWTs are self-contained, all of the necessary information that determines what can be accessed (the scopes) is present. Once the Data Governance Broker and an external resource server are configured with a key pair, the validation of the token and the request for data are processed by the resource server, reducing the need to make multiple requests to the Data Governance Broker.

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.

Note

When a client requests the `offline_access` scope, the server relies on the Offline Access policy to determine if the request included the `prompt` parameter including the `consent` value. If the server finds that `prompt=consent` was not provided, it will remove the `offline_access` scope from the candidate list of scopes to authorize.

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":"VGhlIGFwcGFyaXRpb24gb2YgdGhlc2UgZmFjZXMgaW4gdGhlIGNyb3dkOw==",
  "refresh_token": "UGV0YWxzIG9uIGEgd2V0LCBibGFjayBib3VnaC4=",
  "token_type":"bearer",
  "expires_in":3600,
  "scope": "profile email"
}
```

Accepting External Access Tokens

As a Resource server, the Data Governance Broker supports receipt of access tokens from a third-party OAuth2 provider. Authorization of requests that use externally-defined access tokens can now use a third-party validator.

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 Governance 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.

Data Governance Broker Endpoints for OAuth2 Clients

The Data Governance 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 Governance 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 Governance Broker Endpoints for Clients

Endpoint	Description
/scim	
/scim//v2/<name>	This is the SCIM 2.0 protocol endpoint used to retrieve a specified SCIM Resource Type, where `<name>` 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 `resourceOperations` value that represents the desired action.
/oauth	
/oauth/authorize	The OAuth2 standard authorization endpoint. This is the endpoint that an application will use to get an authorization grant from the user.
/oauth/token	The OAuth2 token endpoint. This is the endpoint that an application will use to request an access token from the Data Governance Broker Server to access identity information.
/oauth/revoke	The Data Governance Broker endpoint used to revoke a token.
/oauth/validate	The Data Governance Broker endpoint used to validate a token.

Endpoint	Description
/userinfo	
/userinfo	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 `openid` 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	
/pdp/v1/authorization	The Data Governance Broker Policy Decision Point endpoint used by an external Policy Enforcement Point (PEP) to generate XACML requests and send them directly to the Data Governance 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.

Chapter 6: Configuring Scopes and XACML Policies

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 Governance Broker before a client can include them in requests. Scopes are also used to capture consent for the requested resources.

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. Default policies are available, or custom policies can be written.

Topics include:

OAuth2 Overview

The Data Governance 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 Governance Broker is installed, the Identity Provider Service can be configured with the `dsconfig` tool or through the Administrative 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 Governance 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.

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

Generic OAth2 Scope Properties

Property	Description
Token Name	The scope name as presented in an OAuth2 request.
Type	The scope type, which is `oauth2` for generic scopes.

Generic OAth2 Scope Properties

Property	Description
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, if the scope is configured to require consent.
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. User attributes can also be obtained as claims in the ID token.

Properties in this scope include those in the generic 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 `authenticated-identity` for authenticated identity scopes.
Resource Operations	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>`, `/Userinfo`, or `/Me` • `replace` (PUT) to endpoint `/scim/v2/<id>` • `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
Resource Attributes	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.
SCIM Sub Resource Types	The Sub Resource Type that can be accessed by this scope, such as consent history, account state, or password criteria.

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. Policy processing of requests that contain account, or external identity provider actions is described in SCIM Sub-Resource Operation Policy Evaluation.

Resource Scope Properties

Property	Description
Type	The scope type, which is `resource` for resource scopes.
SCIM Resource Type	The SCIM Resource Type that can be accessed with this scope.
Resource Operations	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>` • `replace` (PUT) to endpoint `/scim/v2/<id>` • `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
Resource Attributes	A list of one or more SCIM attributes of the SCIM Resource Type for which this scope allows access. The type of access is determined by the operation properties `create`, `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.
SCIM Sub Resource Types	The Sub Resource Type that can be accessed by this scope, such as consent history, account state, or password criteria. If this is null, access is for the base resource object. If not null, access is granted for the sub resource only, and not for the base resource.

For granting access to Data Governance 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).

Scope Authorization Processing

After authentication processing is performed and an ACR is satisfied, each scope requested by the client is evaluated to determine if it can be granted in the final access token. When an OAuth2 client sends an authorization request to the Data Governance Broker's authorize endpoint, the Data Governance Broker first uses XACML policies to drive the authentication flow. The Data Governance Broker will again use XACML policies to authorize the scopes to grant. Finally, an authorization code and/or access token is granted.

Each scope associated with an OAuth2 client can be configured with:

- Whether the scope must be granted when requested by the client. If so and the scope can not be granted for any reason, the entire request will fail.
- Whether the scope requires consent from the end-user when requested by the client.
- The ACR required to grant the scope when requested by the client.

Satisfy Authentication Context Requirements (ACRs)

In the authentication processing phase, the Data Governance Broker's authentication flow is determined using the ACR request parameter from the client. If the Authentication Context satisfies the ACR as is, the policy returns permit. If the Authentication Context can not currently satisfy the ACR as is, it may return a deny with advices to perform authentication flows. Lastly, if the Authentication Context can not satisfy the ACR at all, it may return a deny without any advices.However, there may be cases where the authentication flow should be determined based on the scopes requested. For example, if the client requested a scope that includes access to highly sensitive data, second factor authentication may be desirable to protect the data, even if the client did not specify an ACR to require it.

If specified by configuration, the required ACR of each requested scope is evaluated first, just like during the authentication processing phase. If the Authentication Context satisfies the ACR as is, the policy returns permit. If the Authentication Context can not currently satisfy the ACR as is, it may return a deny with advices to perform authentication flows. Lastly, if the Authentication Context can not satisfy the ACR at all, it may return a deny without any advices. Additional authentication flows are executed, if indicated by the ACR evaluation result.

Note

The same authentication flow should not be executed twice. An ACR policy should check the last time the authentication flow was performed, and only prompt if required. However, a policy could be written to trigger an authentication flow without first performing the check, and cause the same authentication flow to be triggered twice. In that case, the Data Governance Broker will log an error instead of prompting the user again.

Prompt for and Capture Consent

When a scope is granted by policy and consent is required, the previous consent decision for the scope is checked first. If consent was previously granted or denied, it will not be prompted for again. If there was no previous consent decision for the scope, consent approval will be requested. The consent approval prompt will ONLY include scopes that had no previous consent decisions.

When approving the consent approval prompt, all required scopes will be recorded with an approved consent decision. Optional scopes must be explicitly approved or they will be recorded with a deny consent decision.

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 Governance 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 Administrative 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 resource operations allowed by this scope.
- 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 sub resource 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.

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.

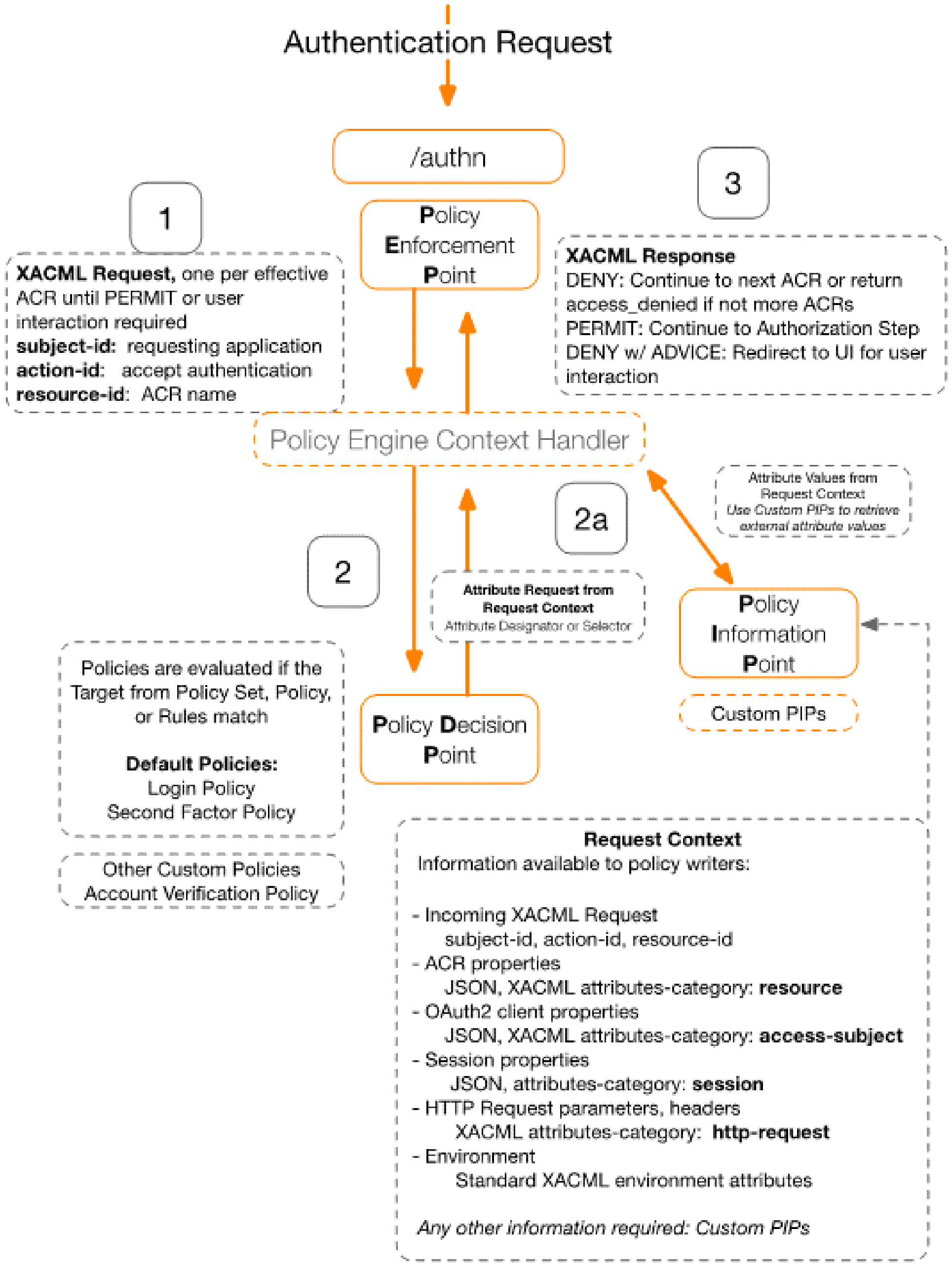
XACML policies are evaluated by the Data Governance Broker in response to the following requests made by OAuth2 clients:

- An authentication (OpenID Connect) request to the OAuth2 endpoint.
- 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.

Authenticating the End-user Prior to Granting an Access Token

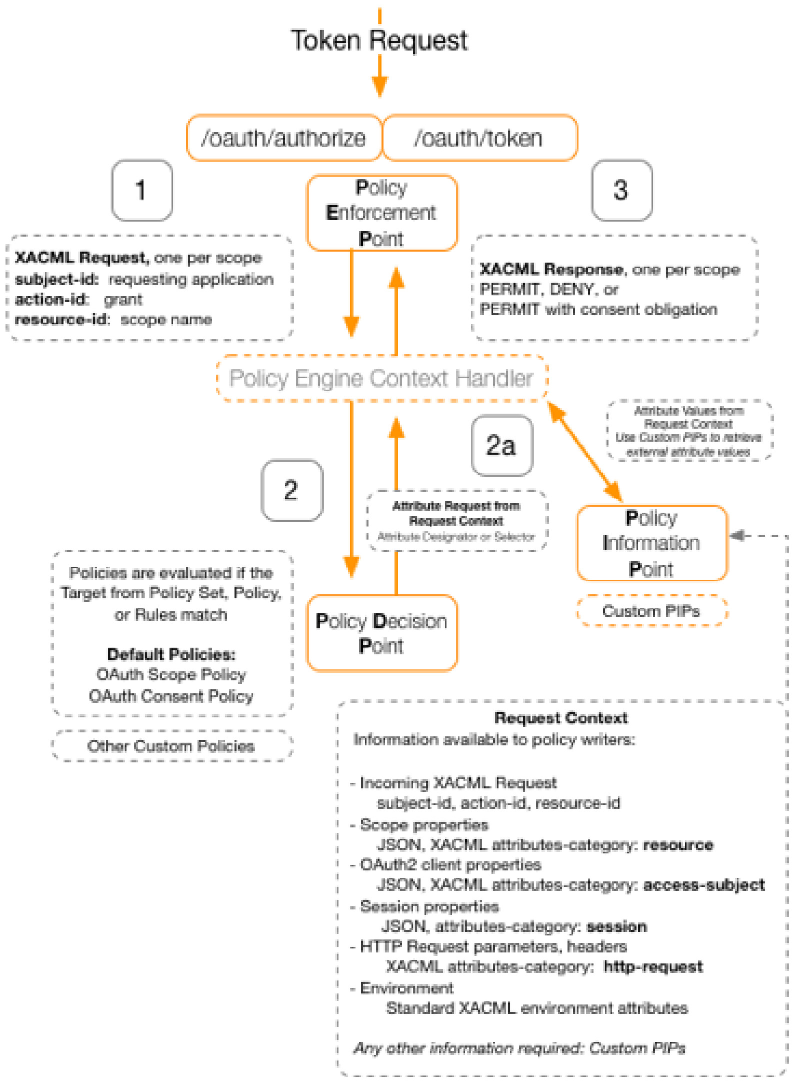
When an OAuth2 client sends an OAuth2 or OpenID Connect authorization request to the Data Governance Broker's authorize endpoint, XACML policies are used to drive the authentication flow.


Authentication Request
/authn
Policy Enforcement Point
1
XACML Request, one per effective ACR until PERMIT or user interaction required
subject-id: requesting application
action-id: accept authentication
resource-id: ACR name
3
XACML Response
DENY: Continue to next ACR or return access_denied if not more ACRs
PERMIT: Continue to Authorization Step
DENY w/ ADVICE: Redirect to UI for user interaction
Policy Engine Context Handler
Attribute Values from Request Context
Use Custom PIPs to retrieve external attribute values
2a
2
Attribute Request from Request Context
Attribute Designator or Selector
Policy Information Point
Custom PIPs
Policies are evaluated if the Target from Policy Set, Policy, or Rules match
Default Policies:
Login Policy
Second Factor Policy
Other Custom Policies
Account Verification Policy
Policy Decision Point
Request Context
Information available to policy writers:
- Incoming XACML Request
subject-id, action-id, resource-id
- ACR properties
JSON, XACML attributes-category: **resource**
- OAuth2 client properties
JSON, XACML attributes-category: **access-subject**
- Session properties
JSON, attributes-category: **session**
- HTTP Request parameters, headers
XACML attributes-category: **http-request**
- Environment
Standard XACML environment attributes
Any other information required: Custom PIPs

Requesting an Access Token

An OAuth2 client requests an access token, receives the token from the Data Governance Broker or a third-party, and then sends the token to the Data Governance 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).

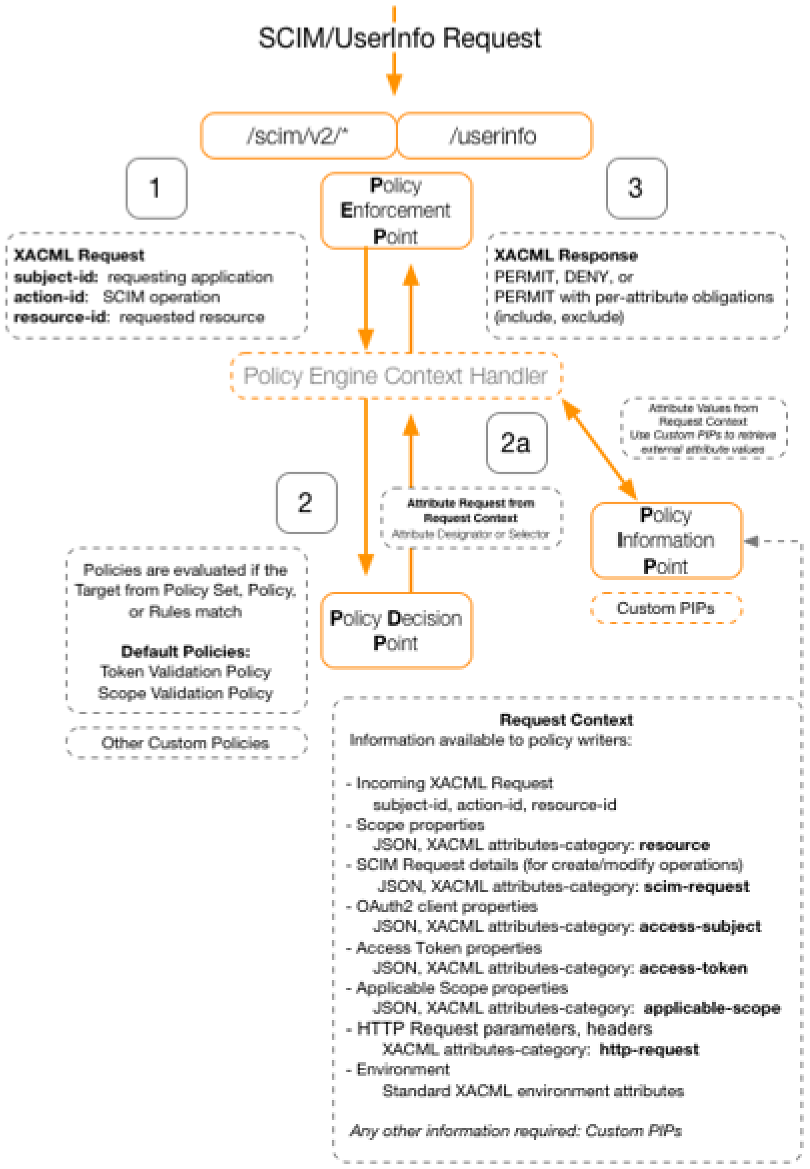
The following illustrates the policy flow for a token request.


Token Request
/oauth/authorize
/oauth/token
1
Policy Enforcement Point
3
XACML Request, one per scope
subject-id: requesting application
action-id: grant
resource-id: scope name
XACML Response, one per scope
PERMIT, DENY, or
PERMIT with consent obligation
Policy Engine Context Handler
Attribute Values from Request Context
Use Custom PIPs to retrieve external attribute values
2a
2
Attribute Request from Request Context
Attribute Designator or Selector
Policy Information Point
Custom PIPs
Policies are evaluated if the Target from Policy Set, Policy, or Rules match
Default Policies:
OAuth Scope Policy
OAuth Consent Policy
Other Custom Policies
Policy Decision Point
Request Context
Information available to policy writers:
- Incoming XACML Request
subject-id, action-id, resource-id
- Scope properties
JSON, XACML attributes-category: resource
- OAuth2 client properties
JSON, XACML attributes-category: access-subject
- Session properties
JSON, attributes-category: session
- HTTP Request parameters, headers
XACML attributes-category: http-request
- Environment
Standard XACML environment attributes
Any other information required: Custom PIPs

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 Governance 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.


SCIM/UserInfo Request
/scim/v2/*
/userinfo
1
Policy Enforcement Point
3
XACML Request
subject-id: requesting application
action-id: SCIM operation
resource-id: requested resource
XACML Response
PERMIT, DENY, or
PERMIT with per-attribute obligations (include, exclude)
Policy Engine Context Handler
2a
Attribute Values from Request Context
Use Custom PIPs to retrieve external attribute values
2
Attribute Request from Request Context
Attribute Designator or Selector
Policy Information Point
Custom PIPs
Policies are evaluated if the Target from Policy Set, Policy, or Rules match
Default Policies:
Token Validation Policy
Scope Validation Policy
Other Custom Policies
Policy Decision Point
Request Context
Information available to policy writers:
- Incoming XACML Request
subject-id, action-id, resource-id
- Scope properties
JSON, XACML attributes-category: resource
- SCIM Request details (for create/modify operations)
JSON, XACML attributes-category: scim-request
- OAuth2 client properties
JSON, XACML attributes-category: access-subject
- Access Token properties
JSON, XACML attributes-category: access-token
- Applicable Scope properties
JSON, XACML attributes-category: applicable-scope
- HTTP Request parameters, headers
XACML attributes-category: http-request
- Environment
Standard XACML environment attributes
Any other information required: Custom PIPs

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 Governance Broker. By default, the combining algorithm for Data Governance Broker policies is `deny-overrides`. This can be changed in the Policy Service through the Administrative 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 Governance Broker provides the following obligation types.

Authentication Requests

Login Required Advice – Indicates that the user must login (again) in order for the requested ACR to be satisfied

Second Factor Required Advice – Indicates that a second factor authentication step is required in order for the requested ACR to be satisfied

Account Flow Required Advice – Indicates that an arbitrary Account Flow must be completed in order for the requested ACR to be satisfied. Takes a single argument "account-flow-handler" which identifies which account flow should be executed.

The following example shows an Account Flow Required advice expression in which the requested flow is Verify Account:

```
<AdviceExpressions>
  <AdviceExpression AdviceId="account-flow-required" AppliesTo="Deny">
    <AttributeAssignmentExpression AttributeId="account-flow-handler">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
       Verify Account
      </AttributeValue>
    </AttributeAssignmentExpression>
  </AdviceExpression>
</AdviceExpressions>
```

OAuth2 Authorization Requests

Override Consent Obligation – By default consent is required or not based on the `permitted-scope` configuration of the requesting OAuth2 client. This obligation can be used to override the value in the configuration. The `isRequired` argument determines whether consent is required or not required.

The following is XACML syntax for a sample consent obligation:

```
<ObligationExpressions>
  <ObligationExpression ObligationId="override-consent" FulfillOn="Permit">
    <AttributeAssignmentExpression AttributeId="is-required">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
       true
      </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 Governance 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 Governance 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. Before authorizing scopes, the Data Governance Broker first determines if the current Authentication Context satisfies the effective ACRs for the request, and if not, what authentication actions must be taken. To do this, the authorization endpoint consults the XACML policies that enforce authentication rules. Each policy request is to determine whether the current Authentication Context (which is mostly defined by the user's session data) meets the requirements specified by a particular ACR.

To differentiate an authentication request from authorization requests and resource requests, the Data Governance Broker uses the XACML action type "accept-authentication." Each authentication policy request generated from the authorization endpoint contains the following information:

OAuth2 Authentication Request Attributes

Attribute ID	Attribute Category	Value
`subject-id`	access-subject	The client name.
`action-id`	accept-authentication	`grant`.
`resource-id`	resource	The requested ACR name.
`<JSON content>`	resource	ACR properties.
`<JSON content>`	access-subject	The OAuth2 client properties.
`<JSON content>`	session	The session properties including the authenticated user resource.

The authorization endpoint asks for an independent policy decision for each scope requested. If the policy decision for any scope is `deny`, the Authentication 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 Governance 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 Governance 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 one of `authorization-code`, `implicit`, `password`, or `client-credentials`.
`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`.

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.

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 Sub-Resource Operation Policy Evaluation

TheDirectory Server can return account status, password restrictions, consent records, consent history, sessions, 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 Directory Server's Password Policy State Extended Operation configuration. See the *PingData Directory Server 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>/consentHistory` – Processes requests to retrieve a user's consent history.
- `/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 Directory Server's default password policy.
- `/scim/v2/<scim-resource-type>/<id>/sessions` – Processes requests to retrieve or revoke a user's session.

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`.

SCIM Search Request Attributes

Attribute ID/Content	Attribute Category	Attribute Value
`resource-id`	resource	Relative URL of the SCIM endpoint, such as `Users`.
`<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 or sub resource to retrieve, such as `Users/12345` or `/Users/12345/consents`.
`<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
`subject-id`	access-subject	The client application name.
`action-id`	action	`create`.
`resource-id`	resource	Relative URL of the SCIM Resource Type to be created, such as `Users` or of the SCIM sub resource to be created, such as `/Users/12345/consents`.
`<JSON Content>`	scim-request	SCIM request body of the resource or sub resource to be created.
`<JSON Content>`	access-token	Access token properties.
`<JSON Content>`	applicable-scope	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
`subject-id`	access-subject	The client application name.
`action-id`	action	`modify`.
`resource-id`	resource	Relative URL of the resource or sub resource to be modified, such as `Users/12345` or `/Users/12345/account`.
`<JSON Content>`	scim-request	The normalized SCIM PATCH request body.

SCIM PATCH Request Attributes

Attribute ID	Attribute Category	Value
`<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	`delete`.
`resource-id`	resource	Relative URL of the resource or sub-resource to be deleted, such as `Users/12345` or `/Users/12345/account`.
`<JSON Content>`	access-token	Access token properties.
`<JSON Content>`	applicable-scope	Applicable scope objects.

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 Governance Broker for evaluation. The request is passed directly to the policy engine. The request can contain any standard XACML attributes, Data Governance 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:pingidentity: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:pingidentity: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 PingData-defined `AttributeDescriptors`. See HTTP Request Properties for details.
- `urn:pingidentity:names:2.0:attribute-category:scim-request` – This custom category is populated by the Data Governance 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:pingidentity: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:pingidentity:names:2.0:session` – This custom category provides access to properties of the current Data Governance 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 Governance 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 Governance 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 Governance 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 `grant` for OAuth2 requests, and will correspond to one of the scope operations.
urn:oasis:names:tc:xacml:1.0:	urn:oasis:names:tc:xacml:3.0:	time	The time at which the Data

Attribute URN	Attribute Category	XACML Data Type	Description
environment:current-time	attribute-category:environment		Governance 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 Governance Broker began processing the current authorization request.

Custom XACML Function

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

The `urn:pingidentity: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 Governance 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.

ACR Properties

When evaluating whether the current Authentication Context meets the requirements of an ACR, the resource category content is an ACR object. Policies that evaluate whether the ACR is satisfied will usually compare values found in the user's current session with the ACR's requirements.

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 Governance Broker will decide which scopes to grant in the access token.

By default, the Data Governance 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 requet 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 &quot;home&quot;].region"
     DataType="http://www.w3.org/2001/XMLSchema#String"/>
```

Accessing Referenced SCIM Resource Attributes

The Data Governance 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 Governance Broker, may be accessed using an `AttributeSelector`, including the following properties.

OAuth2 Client Properties

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

ACR Properties

Policies that evaluate whether the ACR is satisfied will usually compare values found in the user's current session with the ACR's requirements. The following ACR properties are evaluated.

ACR Properties

Property	Data Type	Description
`loginExpirationInterval`	Duration.	The length of time since the last login before the user is required to login again.
`loginAuthenticationChain`	String.	The login authentication chain, if any, that must be used to satisfy the ACR .
`secondFactorExpirationInterval`	Duration.	The length of time since the last second-factor authentication before the user is required to enter a second factor again.
`secondFactorAuthenticationChain`	String.	The second factor authentication chain, if any, that must be used to satisfy the ACR.

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 `authenticated-identity` for authenticated identity scopes, `resource` for resource scopes, or `oauth2` 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 `resource` scope, the SCIM Resource Type that can be accessed.
`resourceOperations`	Multivalued list. Optional.	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>` • `replace` (PUT) to endpoint `/scim/v2/<id>` • `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
`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.
`scimSubResourceType`	Multivalued string.	The Sub Resource Type that can be accessed by this scope, such as consent history, account state, or password criteria.

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:pingidentity:names:2.0:attribute-category:http-request`.

HTTP header values can be obtained using an `AttributeDesignator` with `AttributeId` of the form `urn:pingidentity: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:pingidentity:names:2.0:attribute-category:http-request"
    AttributeId="urn:pingidentity: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:pingidentity: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:pingidentity:names:2.0:attribute-category:http-request"
    AttributeId="urn:pingidentity: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:pingidentity: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:pingidentity:names:2.0:attribute-category:scim-request"
    AttributeId="urn:pingidentity: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 Governance Broker can contain many scopes, only some of which are applicable to the current request. The Data Governance Broker's PIP exposes the applicable scopes under the XACML attribute category `urn:pingidentity: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 Governance Broker.

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

```
<AttributeSelector
    Category="urn:pingidentity: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:pingidentity: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
`sub`	String	Relative SCIM path to the authenticated user resource, such as `Users/123456789`.
`subResource`	JSON object	The SCIM resource object for the token owner.
`lastLoginMethods`	Multi-valued string	A list of login methods used at the last login.
`lastSecondFactorMethods`	Multi-valued string	A list of second-factor methods used at the last second-factor authentication (may be empty).
`lastLoginTime`	DateTime	The time of the user's last login.
`lastIpAddress`	String	The IP address of the user agent for the most recent authentication event.
`lastUAString`	String	The user agent string presented by the user agent for the most recent authentication event.
`lastLoginChain`	String	The authentication chain used for the last login.

Access Token Properties

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

The following properties are available in the access token category.

Access Token Properties

Property	Data Type	Description
`active`	Boolean. Required.	`True` if the token is valid, `false` if token is invalid or has expired.
`client_id`	String.	The ID of the OAuth2 Client to which this token is granted.
`sub`	String.	The unique identifier for the token owner. For user tokens, this will be the relative SCIM path to the user resource, such as `Users/123456789`. For Client Credentials (`app`) tokens, this property is not present.
`subResource`	JSON object.	The SCIM resource object for the token owner. This is not present for `app` tokens.

Access Token Properties

Property	Data Type	Description
`scope`	Multivalued string.	A list of scope names granted by this token.
`app`	String.	The name of the OAuth2 client for which this token was created. For application tokens, this value will be equal to `sub`.
`iat`	DateTime.	The date and time at which the token was created.
`exp`	DateTime.	The date and time at which the token will expire.
`jti`	String.	The unique token identifier.
`token_type`	String.	The type of token.
`username`	String.	The user name of the token owner (not present for application tokens).

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:pingidentity: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 Governance 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

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
     PolicyId="urn:pingidentity:policy:ScopeValidationPolicy"
     Version="1"
     RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <Description>
     Authorizes requests based on the scopes granted by the provided access token.
  </Description>
  <Target/>
  <!-- The XACML action-id (requested operation) -->
  <VariableDefinition VariableId="action-id">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
      <AttributeDesignator
         MustBePresent="true"
         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
         DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </Apply>
   </VariableDefinition>
   <!-- whether the granted scope(s) permit access to all resource attributes -->
   <VariableDefinition VariableId="allAttributesAllowed">
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
         *
       </AttributeValue>
       <AttributeSelector Category="urn:pingidentity:names:2.0:attribute-category:applicable-scope"
                          Path="scope.resourceAttributes"
                          DataType="http://www.w3.org/2001/XMLSchema#string"
                          MustBePresent="false"/>
      </Apply>
    </VariableDefinition>
    <Rule RuleId="urn:pingidentity:rule:ApplicableScope" Effect="Deny">
      <Description>
        Deny access if the requested action is not allowed by any scope in the access
        token.
      </Description>
      <Condition>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
            <VariableReference VariableId="action-id"/>
            <AttributeSelector Category="urn:pingidentity:names:2.0:attribute-category:applicable-scope"
                               Path="scope.resourceOperations"
                               DataType="http://www.w3.org/2001/XMLSchema#string"
                               MustBePresent="false"/>
          </Apply>
        </Apply>
      </Condition>
      <AdviceExpressions>
        <AdviceExpression AdviceId="request-denied-reason" AppliesTo="Deny">
          <AttributeAssignmentExpression AttributeId="error">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              insufficient_scope
          </AttributeValue>
        </AttributeAssignmentExpression>
        <AttributeAssignmentExpression AttributeId="error-description">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            Requested operation not allowed by the granted OAuth2 scopes.
          </AttributeValue>
        </AttributeAssignmentExpression>
      </AdviceExpression>
    </AdviceExpressions>
  </Rule>
  <Rule RuleId="urn:pingidentity:rule:AllowOnlyScopedAttributes" Effect="Deny">
    <Description>
      For create and modify operations, deny if the request impacts attributes
      that are not allowed by the applicable scopes.
    </Description>
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
          <VariableReference VariableId="action-id"/>
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              create
            </AttributeValue>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              modify
            </AttributeValue>
          </Apply>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
          <VariableReference VariableId="allAttributesAllowed"/>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
          <Apply FunctionId="urn:pingidentity:names:2.0:function:scimAttribute-subset">
            <AttributeDesignator Category="urn:pingidentity:names:2.0:attribute-category:scim-request"
                                 AttributeId="urn:pingidentity:names:2.0:impacted-attributes"
                                 DataType="http://www.w3.org/2001/XMLSchema#string"
                                 MustBePresent="false"/>
            <AttributeSelector Category="urn:pingidentity:names:2.0:attribute-category:applicable-scope"
                               Path="scope.resourceAttributes"
                               DataType="http://www.w3.org/2001/XMLSchema#string"
                               MustBePresent="false"/>
          </Apply>
        </Apply>
      </Apply>
    </Condition>
    <AdviceExpressions>
      <AdviceExpression AdviceId="request-denied-reason" AppliesTo="Deny">
        <AttributeAssignmentExpression AttributeId="error">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            insufficient_scope
          </AttributeValue>
        </AttributeAssignmentExpression>
        <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:pingidentity: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:pingidentity: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 Governance 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 Administrative 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.
- [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 Managment 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 Governance 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 Governance Broker provides the following Policy Information Providers:

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

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

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 Governance Broker instance. PIPs with a smaller value are evaluated first to determine if they match a XACML attribute ID.

Creating XACML Policies

The Administrative 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 Governance 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:

Account Verification – Determines whether the current user's account must be verified before authentication can continue. This policy is only installed if `create-initial-broker-config` was run after installation and the option to use starter schemas was chosen. Account verification relies on policy. If starter schemas were not installed, an Account Verification policy will need to be created and added to the Verify Account Flow handler.

Authentication Policy Set – A container for policies that apply to Authentication Context Class evaluations. The combining algorithm for policies is ordered-deny-overrides, meaning that each policy referenced below is evaluated in order. If any policy returns DENY, policies below that are not evaluated.

Login – Determines whether the current session meets the login requirements of an Authentication Context Class Reference (ACR).

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 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 Governance Broker to request the scope. This policy includes a rule that will only permit a user to obtain a Resource scope if the user has a privileged entitlement. The rule checks for a value of `admin` in the user’s `entitlements` attribute. If the user’s schema contains no such attribute then the policy will always deny Resource scopes to that user.

Offline Access – Enforces the OpenID Connect restrictions on permitting offline access to a client requesting a refresh token.

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.

Second Factor Authentication – Determines whether the current session meets the second-factor authentication requirements of an Authentication Context Class Reference (ACR). This policy includes a rule that checks for a boolean attribute `secondFactorEnabled` in the user’s entry. If this attribute does not exist or is set to `false`, then the user will not be able to authenticate with a second factor.

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 Governance 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 Governance Broker.

Note

Policies that are part of a policy set should be disabled in the Data Governance 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 Governance Broker can generate a trace of any policy decision, and log traces with in the File Based Trace Log Publisher with `dsconfig` or through the Administrative Console.

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 `scimType` error parameter.
`request-denied-reason`	`error_description`	The value of the `error_description` 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 Governance 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 Governance Broker only supports one version of a multiple-decision request by using multiple `<Attributes>` of the same category in the request. In addition, Data Governance 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 Governance 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>`
    - `<PolicyCombinerParameters>`
    - `<PolicySetCombinerParameters>`

Chapter 7: Advanced Configuration

The Data Governance Broker's non-user data consists of data in the server configuration. Generally, data in the server configuration define an individual Data Governance 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 Administrative Console or from the `<server-root>/docs/index.html` page. Information in this chapter highlights configuration of interest to a Data Governance Broker installation. For complete configuration options and details, see the Configuration Reference.

Topics include:

General Server Configuration

There are tools and settings that are common across all PingData 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

There are several tools that can be used for server administration and maintenance in the `/bin` directory. The following is a sample of the command-line configuration tools:

Command-line Tools

Tool	Description
backup	Run full or incremental backups on one or more Data Governance 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.
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 Governance Broker configuration.
create-rc-script	Create a Run Control (RC) script that can be used to start, stop, and restart the Data Governance Broker on Unix-based systems.
dsconfig	View and edit the Data Governance 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 Governance Broker and its associated tools. Before launching the command, edit the properties file located in `config/java.properties` to specify the desired JVM arguments and the `JAVA_HOME` environment variable.
encryption-settings	Manage the server encryption settings database.
evaluate-policy	Request a policy decision from the Data Governance Broker.
ldapdelete	Perform an LDAP delete operation.
ldapcompare	Perform an LDAP compare operation.
ldapmodify	Perform LDAP modify, add, and modify DN operations in the Data Governance Broker.
ldappasswordmodify	Perform LDAP password modify operations in the Data Governance Broker.

Command-line Tools

Tool	Description
ldapsearch	Perform LDAP search operations in the Data Governance 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.
list-backends	List the backends and base DNs configured in the Data Governance 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 Governance Broker. Any added extensions require a server re-start.
oauth2-request	Performs OAuth2 requests on the Data Governance Broker. This tool can be used to test OAuth2 functions of the Data Governance Broker, and to manage OAuth2 tokens on behalf of registered applications.
prepare-external-store	Prepares the external Directory Servers for the Data Governance Broker. This is run as part of the `create-initial-broker-config` tool during installation. This tool creates the Data Governance 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 Governance Broker.
remove-defunct-server	Removes a permanently unavailable Data Governance Broker after it has been removed from its topology by the `uninstall` tool.
restore	Restore a backup of the Data Governance Broker.
review-license	Review and/or accept the product license.
server-state	View information about the current state of the Data Governance Broker processes.
start-broker	Start the Data Governance Broker.
status	Display basic server information.
stop-broker	Stop or restart the Data Governance 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 Governance Broker configuration, and is parallel in functionality with the Administrative 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 Governance Broker and use them in the batch file.

Administrative Accounts

Users that authenticate to the Config API or the Administrative Console are stored in `cn=Root DNs,cn=config`. These users must exist on all instances of the Data Governance 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. Accounts can be added or changed with the `dsconfig` tool.

Using the Configuration API

PingData 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:pingidentity: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 `/config/global-configuration` or `/config/backends/userRoot`. Can also list objects when used with a path representing a parent relation, such as `/config/backends`.	`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 `config/backends`.	`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 `/config/backends/userRoot`.	`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 `/config/backends/userRoot`. 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 `/config/backends/userRoot`.	`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:pingidentity: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",
  "dbCheckpointerBytesInterval": "20 mb",
  "dbCheckpointerHighPriority": "false",
  "dbCheckpointerWakeupInterval": "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",
  "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:pingidentity: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:pingidentity: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:pingidentity: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:

  ```
  {
    "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:pingidentity: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:pingidentity: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",
  "dbCheckpointerBytesInterval": "20 mb",
  "dbCheckpointerHighPriority": "false",
  "dbCheckpointerWakeupInterval": "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",
  "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:pingidentity: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
         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 `id eq "userRoot"` or compound filters like `meta.resourceType eq "Local DB Backend"` and `baseDn co "dc=exmple,dc=com"`.
sortBy	Specifies a property value by which to sort.
sortOrder	Specifies either `ascending` or `descending` 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:pingidentity: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",
  "dbCheckpointerBytesInterval": "20 mb",
  "dbCheckpointerHighPriority": "false",
  "dbCheckpointerWakeupInterval": "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",
  "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:pingidentity: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:pingidentity: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 `urn:pingidentity:schemas:configuration:messages:2.0` 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.
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."
}
```

Configuring HTTP Connection Handlers

The server relies on the HTTP connection handler, which relies on one or more servlet extensions. Servlet extensions are responsible for obtaining Java servlets and registering them to be invoked using one or more context paths. For custom servlet extensions created using the Server SDK, the process varies based on using a Java-based or Groovy-scripted extension. See the Server SDK documentation for details.

HTTP connection handlers are responsible for managing the communication with HTTP clients and invoking servlets to process requests from those clients. They can also be used to host web applications on the server. Each HTTP connection handler must be configured with one or more HTTP servlet extensions and zero or more HTTP operation log publishers.

If the HTTP Connection Handler cannot be started (for example, if its associated HTTP Servlet Extension fails to initialize), this does not prevent the entire server from starting. The server's start tool posts any errors to the error log.

The configuration properties available for use with an HTTP connection handler include:

- `listen-address` – Specifies the address on which the connection handler will listen for requests from clients. If not specified, then requests will be accepted on all addresses bound to the system.
- `listen-port` – Specifies the port on which the connection handler will listen for requests from clients. Required.
- `use-ssl` – Indicates whether the connection handler will use SSL/TLS to secure communications with clients (whether it uses HTTPS rather than HTTP). If SSL is enabled, then key-manager-provider and trust-manager-provider values must also be specified.
- `http-servlet-extension` – Specifies the set of servlet extensions that will be enabled for use with the connection handler. You can have multiple HTTP connection handlers (listening on different address/port combinations) with identical or different sets of servlet extensions. At least one servlet extension must be configured.
- `http-operation-log-publisher` – Specifies the set of HTTP operation log publishers that should be used with the connection handler. By default, no HTTP operation log publishers will be used.
- `ssl-cert-nickname` – In scenarios where the multiple public-private key pairs are in a JKS keystore, the LDAPConnectionHandler allows choosing a specific certificate alias through the `ssl-cert-nickname` property. The HTTPConnectionHandler for HTTPS connections should have the same option for parity.
- `key-manager-provider` – Specifies the key manager provider that will be used to obtain the certificate presented to clients if SSL is enabled.

- `trust-manager-provider` – Specifies the trust manager provider that will be used to determine whether to accept any client certificates presented to the server.
- `num-request-handlers` – Specifies the number of threads that should be used to process requests from HTTP clients. These threads are separate from the worker threads used to process other kinds of requests. The default value of zero means the number of threads will be automatically selected based on the number of CPUs available to the JVM.
- `web-application-extension`– Specifies the web applications to be hosted by the server.

For information about other connection handlers, see the *Data Governance Broker Configuration Reference Guide*.

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

Ping Identity 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* (Directory Server and Directory Proxy Servers)
- `ds-auth-allowed-address`: *Adding Operational Attributes that Restrict Authentication* (Directory Server)
- Connection Criteria: *Restricting Server Access Based on Client IP Address* (Directory Server and Directory Proxy Servers)

- Connection Handlers: restrict server access using Connection Handlers (Configuration Reference Guide for all servers)

Problems with SSL Communication

Enable TLS debugging in the server to troubleshoot SSL communication issues:

```
$ dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name com.unboundid.directory.server.extensions.TLSConnectionSecurityProvider \
  --set debug-level:verbose \
  --set include-throwable-cause:true
```

```
$ dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Debug Logger" \
  --set enabled:true \
  --set default-debug-level:disabled
```

In the `java.properties` file, add `-Djavax.net.debug=ssl` to the `start-ds` line, and run `bin/dsjavaproperties` to make the option take effect on a scheduled server restart.

Conditions for Automatic Server Shutdown

All PingData servers will shutdown in an out of memory condition, a low disk space error state, or for running out of file descriptors. The Directory Server will enter lockdown mode on unrecoverable database environment errors, but can be configured to shutdown instead with this setting:

```
$ dsconfig set-global-configuration-prop \
  --set unrecoverable-database-error-mode:initiate-server-shutdown
```

Configuring Traffic Through a Load Balancer

If an Ping Identity server is sitting behind an intermediate HTTP server, such as a load balancer, a reverse proxy, or a cache, it will log incoming requests as originating with the intermediate HTTP server instead of the client that actually sent the request. If the actual client's IP address should be recorded to the trace log, enable `X-Forwarded-*` handling in both the intermediate HTTP server and PingData server. See the product documentation for the device type. For PingData servers:

- Edit the appropriate Connection Handler object (HTTPS or HTTP) and set `use-forwarded-headers` to `true`.
- When `use-forwarded-headers` is set to `true`, the server will use the client IP address and port information in the `X-Forwarded-*` headers instead of the address and port of the entity that's actually sending the request, the load balancer. This client address information will show up in logs where one would normally expect it to show up, such as in the `from` field of the HTTP REQUEST and HTTP RESPONSE messages.

System Alarms, Alerts, and Gauges

PingData servers provide tools to monitor and manage the health of the system. The Data Governance 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 Governance Broker provides a number of alert handler implementations configured with the `dsconfig` tool or the Administrative 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 through 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 '18.583333333333332'.
         :             : 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

PingData 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 Governance Broker.

Viewing and Configuring Log Publishers

Log publishers can be created or modified on each server using the `dsconfig` tool or through the Administrative Console, **Logging, monitoring, and notifications -> Log Publishers**.

Creating a New Log Publisher

PingData provides customization options to create log publishers with the `dsconfig` command or through the Administrative 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

PingData 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

PingData 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
   ```

   ```
   $ 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 validations 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

PingData 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 sever 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 Data Metrics Server
- Using the status Tool

Backend Monitor Entries

Each PingData 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 `connID="15"`, `connectTime="20100308223038Z"`.
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.

Monitoring Components

Component	Description
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	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. The work queue configuration has a `monitor-queue-time` property set to `true` by default. This logs messages for new operations with a `qtime` 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.

Viewing System and Consent data Through the Data Metrics Server

The Data Metrics Server contains several charts to measure and monitor Data Governance Broker system and user consent activity. Charts and data are configured from the Data Metrics Server Server. The following categories can be made available through a Data Metrics Server 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 *PingData Data Metrics Server Administration Guide* for more information.

Using the status Tool

PingData 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 `--maxAlerts` option. For example, status `--maxAlerts 0` suppresses all alerts.

Server SDK Extensions

Custom server extensions can be created with the 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 Server SDK enables creating extensions for the Directory Server, Directory Proxy Server, Data Metrics Server, Data Governance Broker, and Data Sync Server 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 Governance Broker include:

- Policy Information Provider
- Store Adapter

Data Governance Broker Advanced Server Configuration

When a Data Governance Broker is set up from a peer, its server configuration is cloned to the new Data Governance Broker, and the two configurations are linked such that changes to the configuration are applied to both Data Governance Broker servers by default. See Installing a Clone Data Governance 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 Administrative Console.

Configuring Third-Party Store Adapters

Third-party adapters can be created for directory servers, that are not the PingData Directory Server, 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 Governance Broker.
3. Create a SCIM Resource Type schema.
4. Map Store Adapter(s) and SCIM Resource Types using the Administrative 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 Governance Broker (or Directory Server) 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 Administrative Console or with the `dsconfig` 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 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 Governance 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 Governance Broker functions.
Authentication Servlet	Manages requests to the `/authentication` API endpoint (used by the `auth-ui`).
Configuration	Used to enable read and write access to the server's Configuration API.
Documentation	Manages requests for the `/docs` content, which includes the `index.html` page, the generated Configuration Reference Guide, and other product documents.
JWK Servlet	Provides access to the JSON Web Key for token validation.

Servlet	Functions
OAuth2 Servlet	OAuth2 authorization, token, revocation, and validation endpoints.
Policy Decision Point Servlet	XACML PDP endpoint.
SCIM2	Profile access by SCIM Resource Type using SCIM.
UserInfo Servlet	Profile access using OpenID Connect.

Note

Any servlet accepting JavaScript calls from client applications that are hosted at a different location than that of the Data Governance 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 Governance 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
`cors-enabled`	Specifies if the CORS protocol is allowed by the servlet. The default value is `false`.
`cors-allowed-methods`	Specifies the list of HTTP methods allowed for access to resources. The default value is `GET`.
`cors-enable-per-application-origins`	Specifies that a per-application list of allowed origins is consulted. The default value is `false` in the Restrictive policy and `true` in the Per-Application Origins policy.
`cors-allowed-origins`	Specifies a global list of allowed origins. If the `cors-enable-per-application-origins` property is set to `true`, 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.
`cors-exposed-headers`	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.
`cors-allowed-headers`	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 `Origin`, `Accept`, `X-Requested-With`, `Content-Type`, `Access-Control-Request-Method`, and `Access-Control-Request-Headers`.
`cors-preflight-max-age`	Specifies the maximum number of seconds that a preflight request can

Property	Description
	be cached by the client. The default value is `1800` (30 minutes).
`cors-allow-credentials`	Specifies whether requests that include credentials are allowed. This value should be `false` for servlets that use OAuth2 authorization. The default value is `false`.

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.

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.

Public and Private Key Store Configuration

The Data Governance Broker server can be configured to sign access tokens with a private key and expose a public key to enable external resource servers or client applications to read the content of and validate the tokens. If there are multiple Data Governance Brokers in an environment, a key-pair created on one broker will automatically be mirrored on all other brokers. The Data Governance Broker supports RSA key pairs.

A certificate key pair can be created by or imported to the server with the `dsconfig` tool, or through the advanced setting **System -> Key Pairs** in the Administrative Console. For example, the following command can be used to create a new key pair:

```
$ bin/dsconfig -n create-key-pair --pair-name jwt2
```

When a key-pair is created or imported, the private key is encrypted by the preferred encryption settings definition in the encryption settings database and a Certificate Signing Request attribute is created. The private key and Certificate Signing Request are read-only properties, but not the certificate chain. The public key is wrapped in the certificate chain.

The Certificate Signing Request can be taken to a Certificate Signing Authority to obtain a signed, public key certificate. This can then be imported with `dsconfig` to replace the self-signed certificate.

Assign the key pair using the **Access Token Signing Key Pair** property in the Identity Provider Service configuration.

Note

The Data Governance Broker does not automatically rotate expired keys. If using self-signed certificates, reset the `certificate-chain` property when needed. This will regenerate a new self-signed certificate with the specified validity (`self-signed-certificate-validity`). If using signed certificates, renew the certificate (extend its validity) from the Certificate Signing Authority and set the `certificate-chain` property in the key-pair.

Long keys may require more CPU for processing and affect performance, if request volume is high.

Configure Authentication with a SASL External Certificate

By default, the Data Governance Broker authenticates to the Directory Server using LDAP simple authentication (with a bind DN and a password). However, the Data Governance Broker can be configured to use SASL EXTERNAL to authenticate to the Directory Server with a client certificate.

Note

This procedure assumes that Data Governance Broker instances are installed and configured to communicate with the backend Directory Server instances using either SSL or StartTLS.

After the servers are configured, perform the following steps to configure SASL EXTERNAL authentication:

1. Create a JKS keystore that includes a public and private key pair for a certificate that the Data Governance Broker instance(s) will use to authenticate to the Directory Server

instance(s). Run the following command in the instance root of one of the Data Governance Broker instances. When prompted for a keystore password, enter a strong password to protect the certificate. When prompted for the key password, press **ENTER** to use the keystore password to protect the private key:

```
$ keytool -genkeypair \
  -keystore config/broker-user-keystore \
  -storetype JKS \
  -keyalg RSA \
  -keysize 2048 \
  -alias broker-user-cert \
  -dname "cn=Broker User,cn=Root DNs,cn=config" \
  -validity 7300
```

2. Create a `config/broker-user-keystore.pin` file that contains a single line that is the keystore password provided in the previous step.
3. If there are other Data Governance Broker instances in the topology, copy the `broker-user-keystore` and `broker-user-keystore.pin` files into the config directory for all instances.
4. Use the following command to export the public component of the user certificate to a text file:

```
$ keytool -export \
  -keystore config/broker-user-keystore \
  -alias broker-user-cert \
  -file config/broker-user-cert.txt
```

5. Copy the `broker-user-cert.txt` file into the `config` directory of all Directory Server instances. Import that certificate into each server's primary trust store by running the following command from the server root. When prompted for the keystore password, enter the password contained in the `config/truststore.pin` file. When prompted to trust the certificate, enter **yes**.

```
$ keytool -import \
  -keystore config/truststore \
  -alias broker-user-cert \
  -file config/broker-user-cert.txt
```

6. Update the configuration for each Data Governance Broker instance to create a new key manager provider that will obtain its certificate from the `config/broker-user-keystore` file. Run the following `dsconfig` command from the server root:

```
$ dsconfig create-key-manager-provider \
  --provider-name "Broker User Certificate" \
  --type file-based \
  --set enabled:true \
  --set key-store-file:config/broker-user-keystore \
  --set key-store-type:JKS \
  --set key-store-pin-file:config/broker-user-keystore.pin
```

7. Update the configuration for each LDAP external server in each Data Governance Broker instance to use the newly-created key manager provider, and also to use SASL EXTERNAL authentication instead of LDAP simple authentication. Run the following `dsconfig` command:

   ```
   $ dsconfig set-external-server-prop \
     --server-name ds1.example.com:636 \
     --set authentication-method:external \
     --set "key-manager-provider:Broker User Certificate"
   ```

After these changes, the Data Governance Broker should re-establish connections to the LDAP external server and authenticate with SASL EXTERNAL. Verify that the Data Governance Broker is still able to communicate with all backend servers by running the `bin/status` command. All of the servers listed in the "--- LDAP External Servers ---" section should have a status of `Available`. Review the Directory Server access log can to make sure that the BIND RESULT log messages used to authenticate the connections from the Data Governance Broker include `authType="SASL", saslMechanism="EXTERNAL", resultCode=0`, and `authDN="cn=Broker User,cn=Root DNs,cn=config"`.

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 Governance Broker instance in the topology, copy the exported definition and PIN file to the Data Governance 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. Perform the previous steps for all existing key pairs, as private keys will still be encrypted with the previous preferred encryption definition. Delete the existing key pairs and re-import them (which will automatically use the new preferred encryption definition for the private key).

7. After importing the encryption settings definition to all Data Governance Brokers, including the instance 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 Governance Broker servers. See the command line help for the `encryption-settings` tool for arguments.

Note

If the Data Governance Broker's encryption key is compromised, and the Data Governance 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 Governance Broker's REST APIs.

```
$ 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 Governance 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.

Customizing the Authentication User Interface

The Data Governance Broker interface is implemented as a client-side Angular 2 application without a backend server component. It is written using TypeScript and JavaScript. The project's build process leverages `node` and `npm` (like the Administrative Console), and is packaged as a WAR file. See the Angular 2 documentation for more details about tools and customization.

The Data Governance Broker application is deployed as a Web Application Extension with a base-context path of `/auth-ui`. The `auth-ui` source code is shipped with the Data Governance Broker in the `auth-ui-source.tar.gz` file in the `/webapps` directory. This can be extracted into a directory on a development machine for customization. There are additional details included in a readme file.

Most of the `npm` scripts defined in the `auth` project’s `package.json` file are subcommands used by the top-level scripts `dev`, `test` and `prod`.

Example usage:

```
npm run [dev | test | prod]
```

Note

The Data Governance Broker's Authentication API uses a cookie to track user sessions. Cookie management and server domains should be considered when deploying any clients that will use the Authentication API.

The `auth-ui` implementation uses the `/oauth/authorize` and the `/authentication/*` APIs through AJAX to implement the following views and flows:

- Consent prompts
- Error messages
- Login fields and options
- Second-factor authentication
- Recover username or password
- Register new user account
- IDP-callback

Branding

The `auth-ui` interface styling comes from the `assets/css/ubid-account.css` file. To override its styles, either this file can be edited directly, or an additional CSS override file can be added to the project and included in the `copy-assets` script in `package.json`. For example:

1. Add a file called `shopco.css` to the `assets/css` directory.
2. Add the following to the file:

```
.login-div {
  background-color: #222;
}
.login-container a,
.login-container a:hover {
  color: #e15656;
}
```

3. Change the `package.json` file’s `copy-assets` script to include the new file in the CSS by replacing this:

```
cleancss -o ../dist/css/ubid-account.min.css css/ubid-account.css
```

with this:

```
cleancss -o ../dist/css/ubid-account.min.css css/ubid-account.css
css/shopco.css
```

Schema Changes

The `auth-ui` implementation assumes the sample reference schema is being used. To change the reference schema, surface additional attributes, or use another schema, the `auth-ui` project will need to be modified.

The following example adds the `urn:pingidentity:schemas:sample:profile:1.0:birthDate` attribute from the sample reference schema to the registration form:

1. Edit the `app/register/register.html.ts` file in the `auth-ui` project.
2. Add the following after the "Mobile Number" field's `form-group` element:

```
<div class="form-group">
  <label for="birthDate" class="control-label">Birth Date</label>
    <input
[(ngModel)]="resource
['urn:pingidentity:schemas:sample:profile:1.0:birthDate']"
     type="date" class="form-control input-sm" name="birthDate"
     placeholder="Birth Date" tabindex="9">
</div>
```

3. Optionally disable the customization warning message in `app/register/register.component.ts` by replacing this:

```
isExpectedRegistrableAttributes = (registrableAttributes &&
registrableAttributes.length === 5 &&
registrableAttributes.indexOf('userName') !== -1 &&
registrableAttributes.indexOf('name') !== -1 &&
registrableAttributes.indexOf('password') !== -1 &&
registrableAttributes.indexOf('emails[type eq "home"].value') !== -1 &&
registrableAttributes.indexOf('phoneNumbers[type eq "mobile"].value')
!== -1);
```

with this:

```
isExpectedRegistrableAttributes = true;
```

4. On the Data Governance Broker development server, add `birthDate` to the `register-resource-attribute` for the Registration Identity Authenticator with the following `dsconfig` command:

```
$ bin/dsconfig set-identity-authenticator-prop \
  --authenticator-name Registration \
  --add register-resource-
attribute:urn:pingidentity:schemas:sample:profile:1.0:birthDate
```

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 Governance Brokers in a Topology for details.

Note

To remove a server from the topology, it must be uninstalled with the uninstall tool. See Uninstalling theData Governance 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 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.

Updating the Server Instance Listener Certificate

To change the SSL certificate for the server, update the keystore and truststore files with the new certificate. The certificate file must have the new certificate in PEM-encoded format, such as:

```
-----BEGIN CERTIFICATE-----

MIIDKTCCAhGgAwIBAgIEacgGrDANBgkqhkiG9w0BAQsFADBFMR4wHAYDVQQKExVVbmJvdW5kSUQgQ2VydGlmaWNhd
GUxIzAhBgNVBAMTGnZtLW1lZGl1bS03My51bmJvdW5kaWQubGFiMB4XDTE1MTAxMjE1MzU0OFoXDTM1MTAwNzE1Mz
U0OFowRTEeMBwGA1UEChMVVW5ib3VuZElEIENlcnRpZmljYXRlMSMwIQYDVQQDExp2bS1tZWRpdW0tNzMudW5ib3V
uZGlkLmxhYjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKN4tAN3o9Yw6Cr9hivwVDxJqF6+aEi9Ir3W
GFYLSrggRNXsiAOfWkSMWdIC5vyF5OJ9DlIgvHL4OuqP/YNEGzKDkgr6MwtUeVSK14+dCixygJGC0nY7k+f0WSCjt
IHzrmc4WWdrZXmgb+qv9LupS30JG0FXtcbGkYpjaKXIEqMg4ekz3B5cAvE0SQUFyXEdN4rWOn96nVFkb2CstbiPzA
gne2tu7paJ6SGFOW0UF7v018XY1m2WHBIoD0WC8nOVLTG9zFUavaOxtlt1TlhClkI4HRMNg8n2EtSTdQRizKuw9Dd
TXJBb6Kfvnp/nI73VHRyt47wUVueehEDfLtDP8pMCAwEAAaMhMB8wHQYDVR0OBBYEFMrwjWxl2K+yd9+Y65oKn0g5
jITgMA0GCSqGSIb3DQEBCwUAA4IBAQBpsBYodblUGew+HewqtO2i8Wt+vAbt31zM5/kRvo6/+iPEASTvZdCzIBcgl
etxKGKeCQ0GPeHr42+erakiwmGDlUTYrU3LU5pTGTDLuR2IllTT5xlEhCWJGWipW4q3Pl3cX/9m2ffY/JLYDfTJao
JvnXrh7Sg719skkHjWZQgOHXlkPLx5TxFGhAovE1D4qLVRWGohdpWDrIgFh0DVfoyAn1Ws9ICCXdRayajFI4Lc6K1
m6SA5+25Y9nno8BhVPf4q5OW6+UDc8MsLbBsxpwvR6RJ5cv3ypfOriTehJsG+9ZDo7YeqVsTVGwAlW3PiSd9bYP/8
yu9Cy+0MfcWcSeAE
-----END CERTIFICATE-----
```

If clients that already have a secure connection established with this server need to be maintained, information about both certificates can reside in the same file (each with their own begin and end headers and footers).

After the keystore and truststore files are updated, run the following `dsconfig` command to update the server's certificate in the topology registry:

```
$ bin/dsconfig set-server-instance-listener-prop \
  --instance-name <server-instance-name> \
  --listener-name ldap-listener-mirrored-config \
  --set listener-certificate <path-to-new-certificate-file>
```

The `listener-certificate` in the topology registry is like a trust store. The public certificates that it has are automatically trusted by the local server. When the local server attempts a secure LDAP connection to a peer, and the peer presents it with its certificate, the local server will check the `listener-certificate` property for that server in the topology registry. If the property contains the peer server's certificate, the local server will trust the peer. After this trust is established, the handshake is completed using the inter-server certificate.

Index