UnboundID® Identity Broker

Administration Guide

Version 5.1.0

UnboundID Corp

13809 Research Blvd., Suite 500

Austin, Texas 78750

Tel: +1 512.600.7700

Email: support@unboundid.com

Copyright

Table of Contents

Preface

The UnboundID Identity Broker Administration Guide contains procedures to create and manage policies, register applications, and set up resources. It also contains information about management tasks and tools.

About UnboundID

UnboundID Corp is a leading identity infrastructure domain solutions provider with proven experience in large-scale identity data solutions. The Identity Broker is part of the UnboundID Platform. The UnboundID Platform is the consumer-grade identity access and management platform—built specifically to handle the massive scale and real-time demands of hundreds of millions of customers. It delivers a consistent, seamless, personalized brand experience that makes each customer feel valued. The UnboundID Platform provides a unified view of customer data across all applications, channels, partners, and lines of business.

The UnboundID Platform provides the following:

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

- **Support for External APIs** – Standards-based solutions that can interface with various external APIs to access a broad range of services. APIs include XACML 3.0, SCIM, LDAP, OAuth 2.0, and OpenID Connect.

Audience

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

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

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

Documentation

The Identity Broker includes the following documents, available in the `docs` folder of the server.

- *UnboundID Identity Broker Installation Guide (PDF)*
- *UnboundID Identity Broker Administration Guide (PDF)*
- *UnboundID Identity Broker Application Developer Guide (PDF)*
- *UnboundID Identity Broker REST API Reference (HTML)*
- *UnboundID Identity Broker Configuration Reference Guide (HTML)*
- *UnboundID Identity Broker Command Line Reference (HTML)*

Chapter 1: Introduction

Companies need to be able to monetize this valuable user data, while balancing data privacy regulations. The Identity Broker server provides solutions to manage and monitor the authorization and authentication of user data access. This section includes:

Identity Broker Overview

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

The Identity Broker can act as an authorization serverauthorization server, or both an authorization and resource server authorization and resource server.

- As an authorization server, the Identity 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 Identity Broker is designed to make authorization decisions based on dynamic consumer profile and consent data. It is both the policy decision pointpolicy decision point and the OAuth 2.0 provider for externalized authorization. Because the Identity Broker centralizes the policy and consent functions, regulatory and security rules are applied consistently across all applications. In addition, the Identity Broker can be used to create a common identity and single view of the customer through the use of attribute mapping from multiple backend data stores.

Identity Broker Features

The Identity Broker provides the following features for client applications to securely access identity resources:

- **Support for multiple backend data stores**. The Identity Broker supports multiple data stores, with native support for the UnboundID Data Store and extension points for other data stores, such as relational databases. Applications can be written one time for access to the Identity Broker and receive data from any type of infrastructure backend.
- **Authorization based on Policy and Consent**. The Identity Broker ensures that data is provided to only authorized applications. Authorization can be based on industry rules, corporate policy, or consent granted by customers.
- **Unified Data Views**. The Identity Broker provides a way to aggregate attributes from multiple data stores into single views, such as a customer profile view, a subscriber view, or a device view. Data Views specify attribute mapping and renaming across multiple data stores. Applications can provide their end users a unified view of their information based on the Data Views configured.

- **Support for social login**. The Identity Broker can act as a relying party, enabling users to log into client applications and update or create Identity Broker accounts with external identity provider accounts such as Facebook or Google.
- **Standards-based authorization**. The Identity Broker Server provides OAuth 2.0-compliant functionality for token generation, expiration, validation, and revocation. This provides application developers with flexible, secure authorization flows that can be tailored to multiple application types.
- **User interface samples and templates**. The Identity Broker installs a Profile Manager and Sample Sign-In application, if the option is chosen during installation. These applications can be used to demonstrate how a client application makes requests of the Identity Broker for user data, how an end user can grant consent for the application to access that data, and how the Identity Broker returns that data. Identity Broker Server templates can be used for implementing custom user authentication and consent flows.

Identity Broker Console Overview

The Identity Broker Console is an administrative web application designed to manage policies, applications, resources, and the mapping of data from multiple backend data stores. It also provides system and consent metrics, if configured, to surface data gathered from an UnboundID Metrics Engine.

The Identity Broker Console is organized into the following main categories:

- **Privacy Policies** – Specifies what data (resources) is shared with data requestors or applications.
- **Data Requestors** – Specifies the applications that want access to resources and for what purpose.
- **Data Classification** – Specifies the types of resources that requestors may want to access and what action can be taken on a resource. Enables mapping of attributes and configuration of an external identity provider to perform client application authentication.
- **Categories** – Specifies the governance tags and trust levels that are assigned to applications and resources.
- **Metrics** – Displays request and consent data that has been processed by the system, if the UnboundID Metrics Engine is installed.

Review the component diagrams to understand the relationships between each:

Data Stores and Data Views

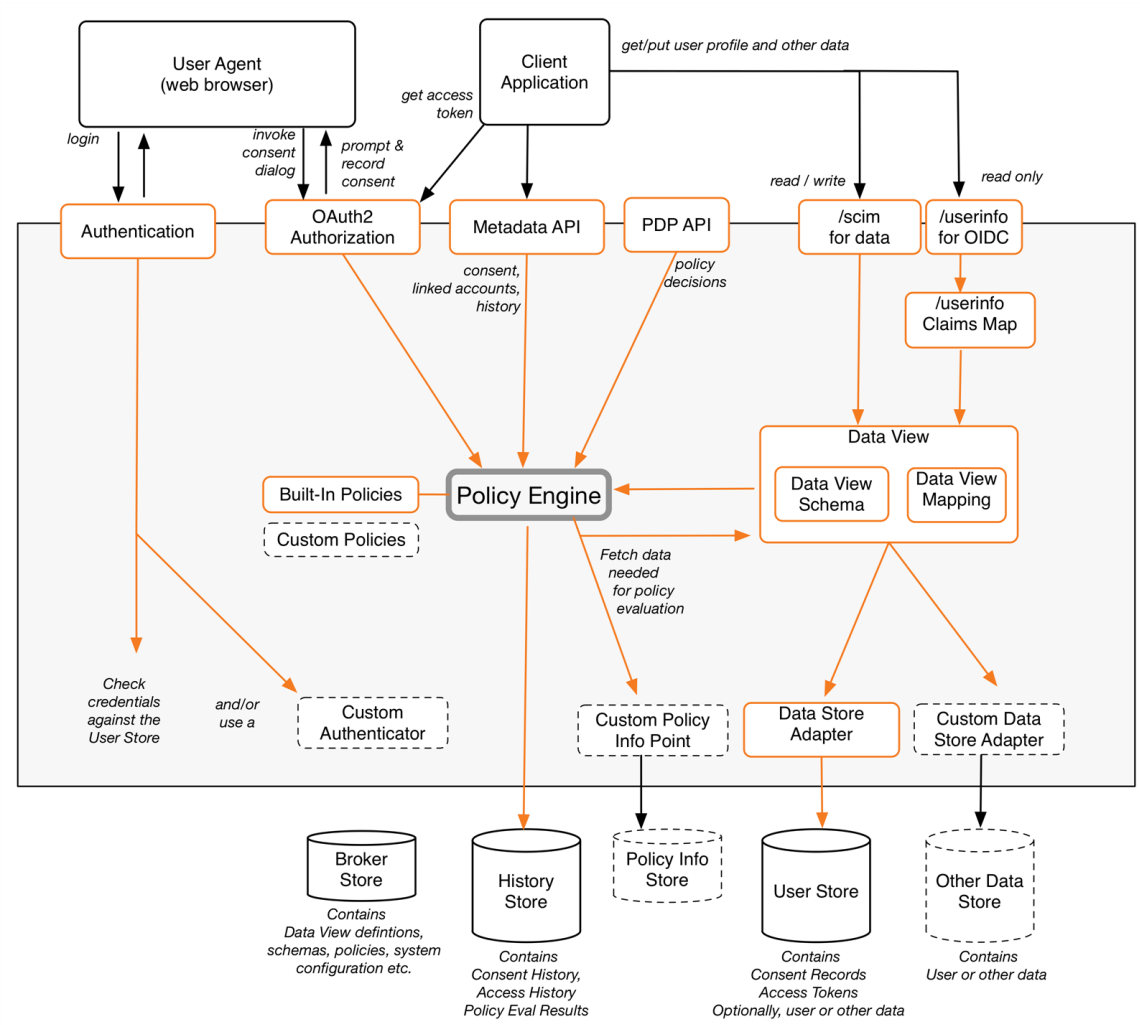
Data Requestors and Data Classification

Policies

Identity Broker Architecture

The Identity Broker can act as both the authorization server and resource server for client applications requesting access to user data. Client applications are granted authorization through an OAuth 2.0 flow and receive access through OpenID Connect and SCIM endpoints.

The Identity Broker can either be an identity provider, or it can be the relying party to an external identity provider, or both. As a relying party, the Identity Broker can offload the authentication responsibilities to a configured identity provider, and use the authenticated principal and any attributes to link end user profiles, or create a new profile in a backend data store.



Identity Broker Architecture

Planning an Identity Broker deployment should start with determining the applications that will request access to data, how they will access the Identity Broker server, and what data can be accessed and updated.

The Policy Engine is key in determining which applications can access resources and for what purpose. Make sure that application development is done with consideration for how policies process requests. See Policies and Request Processing Per Endpoint.

The Identity Broker also tracks the consent that end users grant for access to their data. Consent and access history can be managed by a requesting application or separate application.

Sample Identity Broker Configuration

The following provides a reference sequence of tasks based on the role that the Identity Broker will perform in an existing environment. These tasks can be performed from the Identity Broker Console application or with the `broker-admin` tool. It is recommended that all of the components of an identity infrastructure be identified before beginning any system configuration.

Identity Broker as both a Resource and Authorization Server

The following is a sample workflow for the Identity Broker as both a resource server and authorization server:

1. Determine how user data will be made available to client applications. This includes determining the backend user stores that can be accessed, and how data across multiple stores will be correlated. A store adapter is installed with any LDAP data store. Custom store adapters can be created with the help of UnboundID Professional Services. If mapping attributes from multiple backend data stores, create a Data View Schema for each.
2. Resources can be mapped from multiple data stores to create a unified identity. After Data View Schemas are created and imported into the the Identity Broker Console, resources from each configured user store are available for mapping. Configure Data Views to map resources from multiple backends.
3. If working in an OAuth 2.0 environment, identify the Scopes that can be accessed. Scopes are comprised of Resources, Actions, and optional Purposes. Scopes and actions are required by client applications when sending requests to the Identity Broker.
4. Register the Applications that can request access to data. The application client ID, client secret, scopes, and OAuth 2.0 flows are defined here. This information will be needed by any client application requesting data from the Identity Broker.
5. Determine the Policies and Policy Sets that will govern data access. Policies can base access decisions on the application making the request, the resources requested, the owner of the resource, and the intended action to be taken on the resource. If policies need to access decision-making information outside of the Identity Broker configuration,

a custom Policy Information Provider can be configured with the help of UnboundID Professional Services.

6. Applications and resources can be assigned governance tags and trust levels. These enable enforcing regulatory requirements and provide added security when data is requested from outside sources. These components are optional.
7. Policies determine what and how client applications access resources. To make sure that policy rules work as expected by using Policy Tests and Policy Sandboxes.
8. If using the `/userinfo` endpoint, data must be mapped from backend user stores with User Info Mappings.
9. Client applications can use an external identity provider (Facebook, Google, or OpenID Connect) accounts to access the Identity Broker.
10. If there is an UnboundID Metrics Engine installed, it can be configured to display system and consent metrics for the Identity Broker. See the *UnboundID Identity Broker Installation Guide* for information about configuring the Metrics Engine and Identity Broker server to surface data in the Identity Broker Console.

Identity Broker as an Authorization Server Only

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

1. Identify the resources and data that can be requested by applications. See Resources and Resource Groups.
2. Determine if OAuth2 will be used to authorize access to resources. If so, identify the Scopes that can be accessed, Actions that can be performed, and optional Purposes.
3. Continue with steps 4-10 of the previous workflow.

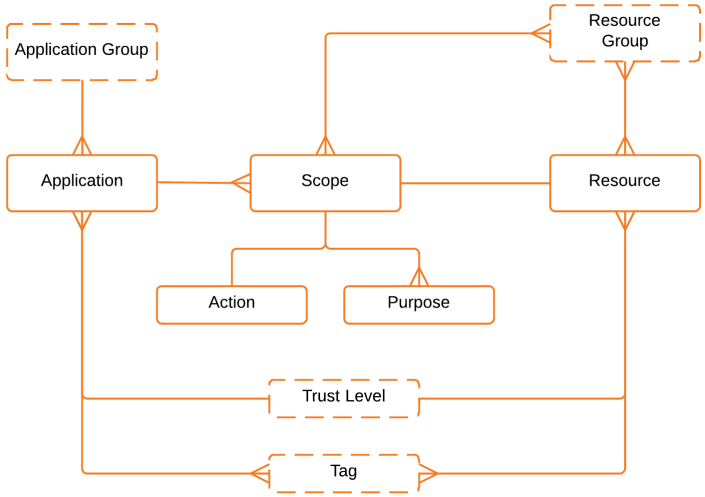
Chapter 2: Data Requestors

Data requestors are the client **Applications** that ask the Identity Broker for access to protected resources. Each request through the Policy Engine must contain an **Action** and can contain a **Purpose**. Actions specify the type of access requested, such as read or write. Purposes are the reason an application is requesting access to a resource, such as Marketing or Billing Verification.

This section explains how to manage data requestor components in the Identity Broker Console application. In this section, the following tasks are performed:

Data Requestors and Data Classification Components

The following illustrates the relationship between the resource components that are configured for the system, the applications that can request access to them, and the actions that can be performed.



Data Requestors and Data Classification Components

These are the components that are used to determine what a client application can request, and what consent flow is needed for access. The security requirements for data access, OAuth 2.0 grant types, and end-user consent should all be determined prior to registering an application with the Identity Broker. Much of this configuration information is needed when configuring or creating client applications that will access Identity Broker resources.

Managing Applications

Create and maintain applications that can request access to resources based on policy, trust level, and any other privacy restrictions. The information used to register the application with the Identity Broker will be needed by the application to request resources. This information includes:

- The client ID and client secret that are generated when the application is registered.
- The OAuth access grant types, token duration, and consent requirements configured for the application.
- Application redirect URIs, which must also be registered with the Identity Broker.
- The scopes that can be requested by an application.
- If an application is given a governance tag or trust level assignment, the assignment must correspond to any tags or trusts assigned to the resources that the application will request. For example, if an application is considered less trusted than a resource, the default Trust Level policy will not allow the application to access that resource. If a resource is tagged as HIPAA, the default Governance Tag policy will only allow applications with the HIPAA tag to access it.
- The trusted origin(s) of the client application if making JavaScript requests. See the *UnboundID Application Developer Guide* for information about how client applications access the Identity Broker resources.
- Any external identity providers that can be used to authenticate an end user account.

To Register a New Application

1. Click **Data Requestors**, and select **Applications**.
2. On the Applications page, click **New Application**.
3. On the **General** tab, enter a name and optional description for the application.
    a. Enter an optional application URL in the **Application URL** box. This can be displayed on an end user's profile page.
    b. Enter an optional URL for the application icon, which displays on the Applications page.
    c. Enter an optional contact email for the application's vendor in the **Contact Email** box. This can be displayed on an end user's profile page.
    d. To assign this application to a group, select one from the **Application Groups** drop-down list.

4. On the **Policy** tab, assign a trust level to the application or assign a tag to identify any governance policies that is applied to the application. An assigned trust level must correspond to trust levels applied to resources that the application may want to access. If an application is considered less trusted than a resource, the application will not be allowed to access to that resource.
5. On the **OAuth** tab, determine how the OAuth 2.0 token will be issued and authorized to a client.
    a. OAuth 2.0 defines different access grant types with different authorization mechanisms. Select the OAuth 2.0 grant type for the application.
    b. Enter the duration for grants, tokens, authorization and consents. In each box, enter **0** for no expiration, or specify a number and unit of time (**s**=seconds, **m**=minutes, **h**=hours, **d**=days, **w**=weeks).
    c. Select the **Override Refresh Token Validity Duration** box if the refresh token is one of the permitted grant types. Enter the duration value.
    d. Choose a signing algorithm from the drop-down list.
6. On the **Redirect URIs** tab, enter the redirect URI for the application.
7. On the **Scopes** tab, add or remove the type of resources that can be accessed. For example, First Name, Last Name, or Email and others.
8. On the **Trusted Origins** tab, add origins to identify trusted sources of client-side JavaScript requests to Identity Broker APIs. This enables Cross-Origin Resource Sharing. This requires additional configuration on the Identity Broker Server.
9. On the **External Identity Providers** tab, choose the external identity providers that this application can use for authentication. These providers are defined in Data Classification and are used to validate existing user credentials with identity providers such as Facebook and Google.
10. Review the settings, and click **Save**.

To Edit an Application

1. Click **Data Requestors**, and select **Applications**.
2. On the Applications page, click the **Edit** button for the application to modify.
3. On the **General** tab, enter a name and optional description for the application.
    a. Enter or change the application URL in the **Application URL** box.
    b. Enter or change a URL for the application icon to display on the Applications page.
    c. Enter or change a contact email for the application's vendor in the **Contact Email** box.

   d. To assign this application to a group, select one from the **Application Groups** drop-down list.

4. On the **Policy** tab, assign a trust level to the application or assign a tag to identify any governance policies that is applied to the application.
5. On the **OAuth** tab, determine how the OAuth 2.0 token will be issued and authorized.
   a. Select the OAuth 2.0 grant types for the token: Web Application, Client Credentials, Implicit, Password.
   b. Enter the duration for grants, tokens, authorization and consents. In each box, enter **0** for no expiration, or specify a number and unit of time (**s**=seconds, **m**=minutes, **h**=hours, **d**=days, **w**=weeks).
   c. Select the **Override Refresh Token Validity Duration** box if the refresh token is one of the permitted grant types. Enter the duration value.
   d. Choose a signing algorithm from the drop-down list.
6. On the **Redirect URIs** tab, enter the redirect URI for the application. If there are more than one for different applications, enter it here.
7. On the **Scopes** tab, add or remove the type of resources that will be accessed. For example, First Name, Last Name, or Email and others.
8. On the **Trusted Origins** tab, add any Application Trusted Origins to identify trusted sources of client-side JavaScript requests to Identity Broker APIs.
9. On the **External Identity Providers** tab, add or remove the external identity providers that this application can use for authentication. These providers are defined in Data Classification.
10. Review the settings, and click **Save**.

To Reset a Client Secret

1. Click **Data Requestors** and select **Applications**.
2. On the Applications page, click the drop-down arrow next to the **Edit** button for the application to modify.
3. Click **Reset client secret**. This resets the OAuth 2.0 application client secret associated with the client ID. The application receives a new, auto-generated secret, which can be viewed in the **OAuth** tab of the Edit dialog.
4. Confirm the action.

To Revoke All Authorizations

1. Click **Data Requestors** and select **Applications**.
2. On the Applications page, click the drop-down arrow next to the **Edit** button for the application to change.
3. Click **Revoke Authorizations**. Any authorizations to access resources are revoked. For OAuth applications, this includes all tokens.
4. Confirm the action.

To Delete an Application

1. Click **Data Requestors** and select **Applications**.
2. On the Applications page, click the drop-down arrow next to the **Edit** button for the application to delete.
3. Click **Delete**.
4. Confirm the action.

To Assign Client Credentials to Resource Servers

An application is assigned a client ID and client secret in the following ways:

- At time of creation: if both grant types and scopes are specified.
- At time of update: if the client secret is reset.

Some client applications, such as resource servers, may need to validate access tokens but do not need to request access tokens. These applications need client credentials but do not need to be assigned grant types or scopes. For resource servers, create an application with no grant types or scopes specified. After creating the app, reset the client secret and use the new client ID and secret listed on the **OAuth** tab of the Edit Application dialog.

Managing Application Groups

Associate applications into groups for easier management. For example, create a "Partners" group that includes applications specific to business partners. Or, create groups for "Internal" and "External" applications. This is optional.

To Create a New Application Group

1. Click **Data Requestors** and select Application Groups.
2. On the Application Groups page, click **New Application Group**.
3. On the General tab, enter a name for the application group in the **Name** box.

4. In the **Description** box, enter a description for the application group.
5. Click the **Applications** tab, and then select the applications to add to this group.
6. Click **Save**.

To Edit an Application Group

1. Click **Data Requestors** and select **Application Groups**.
2. On the Application Groups page, click the **Edit** button for the group to modify.
3. On the **General** tab, change or description of the group.
4. On the **Applications** tab, add or remove applications from this group.
5. Click **Save**.

To Delete an Application Group

1. Click **Data Requestors** and select **Application Groups**.
2. On the Application Groups page, click the drop-down arrow next to the **Edit** button for the group to delete.
3. Select **Delete**.
4. Confirm the action. The group is deleted, but the applications are still available.

Managing Actions

An action identifies the operation the application intends to perform on the specified resources. Typically, these actions are "read" and "write." Add or edit actions that define what can be done with protected resources. Actions are required for defining scopes. Scopes are required for OAuth 2.0 requests.

To Create a New Action

1. Click **Data Requestors** and select **Actions**.
2. On the Actions page, click **New Action**.
3. Enter a name for this action in the **Name** box.
4. Enter an optional description of the action.
5. Click **Save**.

To Edit an Action

1. Click **Data Requestors** and select **Actions**.
2. On the Actions page, click **Edit** for the action to be modified.

3. Change the name or description of the action.
4. Click **Save**.

To Delete an Action

1. Click **Data Requestors** and select **Actions**.
2. On the Actions page, click the drop-down arrow next to the **Edit** button for the action to delete.
3. Select **Delete**.
4. Confirm the action.

Managing Purposes

A purpose identifies the reason why an application is requesting access to a resource. For example, a purpose may represent "Billing Verification" or "Marketing." This is optional.

To Create a New Purpose

1. Click **Data Requestors** and select **Purposes**.
2. On the Purposes page, click **New Purpose**.
3. Enter a name to identify the purpose in the **Name** box.
4. Enter a description.
5. Click **Save**.

To Edit a Purpose

1. Click **Data Requestors** and select **Purposes**.
2. On the Purposes page, click the **Edit** button for the purpose to modify.
3. In the **Name** box, change the purpose name.
4. In the **Description** box, change the purpose description.
5. Click **Save**.

To Delete a Purpose

1. Click **Data Requestors** and select **Purposes**.
2. On the Purposes page, click the drop-down arrow next to the **Edit** button for the purpose to delete.

3. Select **Delete**.
4. Confirm the action.

Chapter 3: Data Classification

Data Classification identifies and manages the data that applications want to access.

This section explains how to manage data classification components in the Identity Broker Console application. In this section, the following tasks are performed:

Managing Resources

Managing Resource Groups

Managing Scopes

Managing Data Views

Managing External Identity Providers

Managing User Info Mappings

Data Classification Components

Data Classification includes the following components:

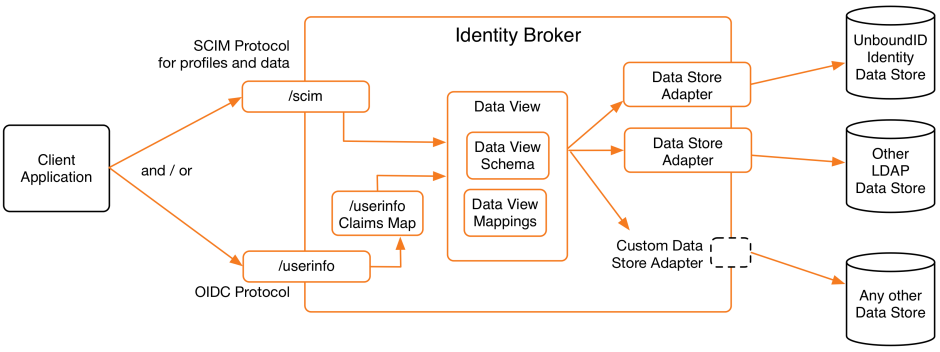
- **Data Views** map attributes in a Data View Schema to native attributes found in data store entries, which provides a unified view of identity data found in multiple data stores. **Data View Schemas** are expressed using the standard SCIM schema format and define the attributes that comprise a Data View. The Data View determines the attributes that can be accessed by a client application.

  Resource groups are created for each Data View. These resource groups consist of all resources created for the attributes from the Data View schema. This group may be used in scopes so client applications can request access to all attributes in the Data View without having to list them individually.

- **Resources and Resource Groups** are the Identity Broker's means of tracking the types of information that can be requested by applications. A Resource uses a URN (Unique Resource Names) to identify the data being requested. A Resource Group is an arbitrary collection of resources that do not necessarily fall within a single portion of the URN hierarchy.
- **Scopes** are defined by OAuth 2.0 as defining a type of resource access. A request from a client application includes an OAuth2 scope which identifies which resources are being requested, such as a user's mailing address and phone number. Scopes are defined within the Identity Broker and are tied to the resources that can be requested, the action that can be taken on that resource, and the optional purpose for the action.
- **External Identity Providers** enable the Identity Broker to authenticate an existing user, or create a new user when that user logs into the Identity Broker through an external identity provider. If configured, the Identity Broker login page will display a Facebook, Google, or other OpenID Connect provider login option. If the end user chooses to login through an external provider, that provider will perform the account authentication. The Identity Broker will still manage the OAuth2 authorization flow. If the account is new to the Identity Broker, and attributes required by the associated Data View are missing from those gathered with the authentication step, the Identity Broker login page will display a registration form to request the missing information from the user.
- **User Info Mappings** map OpenID Connect claims to attributes in the Data View, based on the Open ID Connect standard. This is only required if making requests through the UserInfo endpoint.

Data Stores and Data View Components

The following illustrates the relationship between the backend data stores and the Data Views that can be created to map resources from multiple sources.



Data Mapping Components

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

Public Endpoints: UserInfo and SCIM

The Identity Broker, acting as a resource server, retrieves user profile data through the UserInfo endpoint (`/userinfo`) or provides full read/write access through the SCIM endpoint (`/scim/{name}`). The access to these resources is subject to policy rules and restrictions.

Claims Map (UserInfo Map)

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

Data View Schema

When the Data View schema is imported, the Identity Broker generates resource objects that represent a single attribute or resource. The format for the Data View schema is defined in the SCIM specification. It is created and imported as a JSON file. An example Data View schema is located in the Identity Broker `<server-root>/resource/defaultUserSchema.json`.

Store Adapters

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

Identity Broker provides an LDAP store adapter. Third-party store adapters can be created with Server SDK extensions.

Data View Mappings

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

Data Stores

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

When a store adapter is added to the Identity Broker's server configuration, a correlation attribute can be defined for Data Views that are backed by multiple store adapters. The correlation attribute defines an attribute for each store adapter that is used to uniquely identify the same end user data across different store adapters. For example, if every data store in a User Store stores a user's email address, and an email address can always be considered a primary key (that is, it is always unique per use), then each store adapter's email address attribute can be set as its correlation attribute.

Identity Broker Scopes

Several scopes are installed with the Identity Broker. Each can be configured to best suit administrative needs and the needs of client applications. See Identity Broker Administrative Resources for a listing of all resources that can be used in administrative scopes.

Administrative Scopes

The Identity Broker installs a set of administrative scopes that are required to perform Identity Broker tasks. The Identity Broker Console and the `broker-admin` command line tool are the only applications configured to use these scopes initially. All administrative scopes contain resources with the the `urn:unboundid:resources:broker_admin` prefix. The default Admin API Policy enables access to accounts with the `broker_admin` entitlement for administrative resources, which are grouped in the following scopes. These scopes should be edited with caution.

Admin Token Refresh – Enables an administrator to acquire a refresh token.

Export Configuration – Enables exporting the Identity Broker configuration stored in the Broker Store. This includes the top-level resource for Identity Broker administrative resources, which if granted, enables the requester to access all Identity Broker administrative functions.

Import Configuration – Enables importing the Identity Broker configuration stored in the Broker Store. This includes the top-level resource for Identity Broker administrative resources, which if granted, enables the requester to access all Identity Broker administrative functions.

Invoke PDP – Generally used by third-party resource servers. This is used to invoke the policy decision point without going through the Identity Broker's OAuth 2.0 authentication mechanism.

Read Configuration – Enables reading Identity Broker configuration details for applications, actions, purposes, requests, scopes, resources, tags, trust levels, trace filters, external identity providers, data views, and claims maps.

Read XACML Policies – Enables reading the policies that are configured for the Identity Broker.

Update XACML Policies – Enables create, edit and delete operations on policies configured for the Identity Broker.

Update Configuration – Enables create, edit and delete operations for all Identity Broker functions.

Application Scopes

The rest of the installed scopes are for use by client applications requesting access to Identity Broker resources or functions. They can be modified to best fit the resource requests of specific applications. Make sure that policies are configured to provide or deny access on any new scopes defined.

Billing – A sample scope for a client requesting billing history resources. This is installed if the "Install Sample Data" option was selected during installation.

Consumer Data – A sample scope for a client requesting consumer data and preferences. This is installed if the "Install Sample Data" option was selected during installation.

Delete Profile – Enables an application to delete a user profile.

Manage Consents – Enables an application to perform create, edit, and delete operations on the consents granted for data access.

Manage Links – Enables create, edit, and delete operations on the links that have been created with external identity provider accounts.

Offline Access – Enables an application to cache data and make it available when the requesting application is not online.

openid – The required scope for all OpenID Connect client application requests.

Read Access History – Enables an application to read the granted and revoked consent records for requested user information.

Read Address – Enables an application to read the OpenID Connect address resource for an Identity Broker account.

Read Consents – Enables an application to read the consent information for an Identity Broker account.

Read Email – Enables an application to read the OpenID Connect email resource for an Identity Broker account.

Read Link Authorization – Enables an application to read the authorization credentials for external identity provider accounts that are linked to an Identity Broker account.

Read Links – Enables an application to read the link data, such as external identity provider name, for linked accounts.

Read Phone – Enables an application to read the OpenID Connect phone resource for an Identity Broker account.

Read Profile – Enables an application to read the OpenID Connect profile resource for an Identity Broker account.

Update Address – Enables an application to perform all actions on the OpenID Connect address scope for an Identity Broker account.

Update Email – Enables an application to perform create, edit, and delete actions on the OpenID Connect email resource for an Identity Broker account.

Update Phone – Enables an application to perform create, edit, and delete actions on the OpenID Connect phone resource for an Identity Broker account.

Update Profile – Enables an application to perform create, edit, and delete actions on the OpenID Connect profile resource for an Identity Broker account.

Identity Broker Administrative Resources

The ability to perform operations through the Identity Broker's Admin API, the Console, and with the `broker-admin` tool is determined by policy. By default, the Admin API Policy denies access to any user that does not have the `broker_admin` entitlement or any application that does not request a scope that contains resources with the `urn:unboundid:resources:broker_admin` prefix.

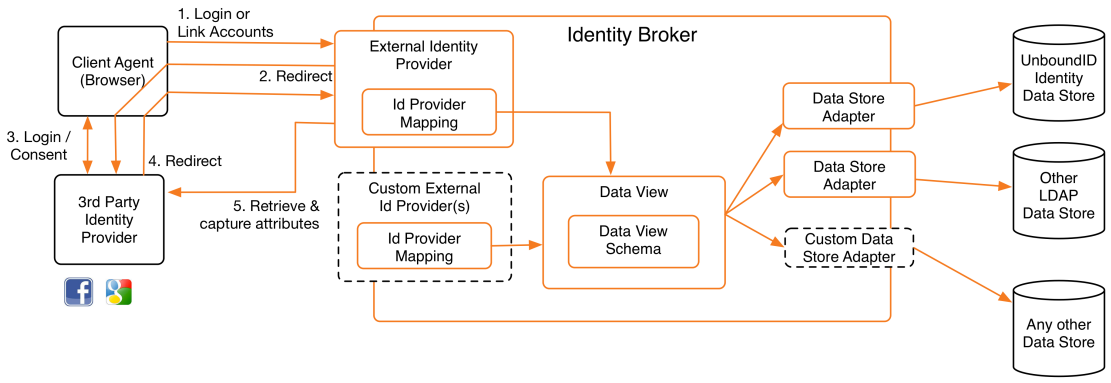
The Identity Broker installs all of the necessary scopes and resources to manage the system. See Identity Broker Scopes for a list of installed scopes. All administrative tasks are defined by scopes that contain the following resources:

- `urn:unboundid:resources:broker_admin` – Top level resource that includes actions for import and export of Broker Store, and for modifying the deletable and editable properties.
- `urn:unboundid:resources:broker_admin:policies` – Includes tasks for Policy, Policy Sandboxes, and Policy Sets.
- `urn:unboundid:resources:broker_admin:applications` – Includes tasks for Applications and Application Groups.
- `urn:unboundid:resources:broker_admin:actions` – Includes tasks for Actions.
- `urn:unboundid:resources:broker_admin:purposes` – Includes tasks for Purposes
- `urn:unboundid:resources:broker_admin:requests` – Includes tasks for Requests

- `urn:unboundid:resources:broker_admin:resources` – Includes tasks for Resources, Resource Aliases, and Resource Groups
- `urn:unboundid:resources:broker_admin:scopes` – Includes tasks for Scopes.
- `urn:unboundid:resources:broker_admin:tags` – Includes tasks for Tags.
- `urn:unboundid:resources:broker_admin:traceFilters` – Includes tasks for managing policy trace filters.
- `urn:unboundid:resources:broker_admin:trustLevels` – Includes tasks for Trust Levels.
- `urn:unboundid:resources:broker_admin:externalIdentityProviders` – Includes tasks for External Identity Providers and managing Identity Provider data.
- `urn:unboundid:resources:broker_admin:dataViews` – Includes tasks for Data Views and managing Store Adapter Data.
- `urn:unboundid:resources:broker_admin:claimsMaps` – Includes tasks for Claims Maps.
- `urn:unboundid:resources:broker_admin:pdp` – Includes tasks for invoking the Policy Decision Point.

The Identity Broker as Relying Party

The Identity Broker, as relying party, acts as a client of an external identity provider service. Users can log into the Identity Broker with external identity provider accounts. The Identity Broker provides authentication claims, account linking, and profile retrieval services to the client application.



Data Flow with an External Identity Provider

The Identity Broker must be registered as an application with the identity provider to enable this flow. External identity providers are configured through the Identity Broker Console External Identity Providers pages or through the `broker-admin` command-line tool.

A social login link (and icon) is displayed on the Identity Broker's default login page for applications configured to use an external identity provider. The login template reads this information through the `LoginPageContextProvider`. See About Velocity Templates for more information.

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

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

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

```
response_type=code
client_id=<relying party application client id>
redirect_uri=https://<rp_host>/metadata/v1/providers/<external identity provider name>/ca
llback
state=<state value generated by the /idpLogin.do endpoint>
scope=<all scopes registered with the relying party application, including 'openid'>
```

After the end user authenticates to the external identity provider and authorizes the OpenID Connect request, the external provider redirects the browser to the Identity Broker's `/idpLogin.do` endpoint, as provided in the `redirect_uri` value. If a matching account is found at the Identity Broker, then the end user will need to log in to link the Identity Broker account and the account at the external provider. Otherwise, a new Identity Broker account can be created.

Note

The `redirect_uri` value used in this flow should be registered as a redirect URI with the application used by the Identity Broker at the external identity provider. It should have the form `https://<identity broker>/idpLogin.do?idp=<idp name>`.

Creating an Account through Identity Provider Login

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

The Identity Broker applies the Data View mappings for the identity provider (configured in the Identity Broker Console, or with the `broker-admin` tool) to the retrieved profile data. If any attribute value required by the Data View is missing, a registration form is displayed to prompt the end user for missing data. The user supplies the information, which is submitted to the SCIM `/registration.do` endpoint with the following parameters. If no additional information is needed, a new Identity Broker account is created.

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

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

Linking Identity Broker and External Identity Provider Accounts

The Identity Broker provides information linking a local account to an external identity provider account through the Metadata REST API at the `/metadata/v1/<userId>/links` endpoint. Client applications can use this API to retrieve or remove an existing link, or to add a new link.

Access to this endpoint is granted to an application by consent to use one of the following links scopes:

Scopes for Linked Accounts

Scope name	Function
`read_links`	Read the links attribute, excluding external IDP credentials.
`read_links_authorizations`	Read external IDP credentials.
`manage_links`	Create, update or delete links.

Data provided by the `/metadata/v1/<userId>/links` endpoint includes:

- `accessToken`
- `expireTime`
- `refreshToken`
- `providerUserId`
- `provider`
    - `name`
    - `type`
    - `description`
    - `iconUri`
    - `userInfoEndpoint` (for OpenID Connect identity providers)

For information about using the `/links` endpoint, see the *Identity Broker REST API Reference* online documentation.

If any external identity provider attributes are mapped to the user's data view, values for these attributes are copied to the user's local profile when logging in through an external identity provider. Applications can also retrieve data from an external identity provider account using data from the `/metadata/v1/<userId>/links` endpoint.

Note

Access to external identity provider data requires consent from the end user.

Example Call for Links Data

If an application has an end user's unique SCIM ID and a bearer token for the `read_links` and `read_link_authorizations` scopes, it can obtain a list of the end user's linked identity

provider accounts, including the account IDs and access tokens needed for limited read access to those accounts.

```
 GET /metadata/v1/9f8a23-a7171c48-fde2-3224-9087-81167f65df2f/links HTTP/1.1
Accept: application/json
Authorization: Bearer VGltZSBwcmVzZW50IGFuZCB0aW1lIHBhc3QgLyBBcmUgYm90aCBwZXJoYXBzIHByZXN
lbnQgaW4gdGltZSBmdXR1cmU=

HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 1,
    "data": [
        {
             "accessToken": "SWYgYWxsIHRpbWUgaXMgZXRlcm5hbGx5IHByZXNlbnQgLyBBbGwgdGltZSBp
cyB1bnJlZGVlbWFibGUu",
             "expireTime": 1414178475000,
             "provider": {
                 "appId": null,
                 "clientSecret": null,
                 "deletable": true,
                 "description": null,
                 "editable": true,
                 "iconUri": "https://<example.com>/icons/facebook_32.png",
                 "id": "DATTA",
                 "modifyTimestamp": null,
                 "name": "Facebook Relying Party App",
                 "permissions": null,
                 "type": "facebook"
             },
             "providerUserId": "26091888",
             "refreshToken": null
         }
    ],
    "startIndex": 0,
    "totalResults": 1
}
```

Based on the `accessToken`, `providerUserId`, and `provider.type` values in the above response, the application can formulate a profile request for the external identity provider. For example, the following is a Facebook Graph API 2.0 request:

```
GET /v2.0/26091888 HTTP/1.1
Accept: application/json
Authorization: Bearer SWYgYWxsIHRpbWUgaXMgZXRlcm5hbGx5IHByZXNlbnQgLyBBbGwgdGltZSBpcyB1bnJ
lZGVlbWFibGUu
Host: graph.facebook.com

HTTP/1.1 200 OK
Content-Type: application/json

{
   "email": "tom.eliot@example.com",
   "first_name": "Tom",
   "gender": "male",
   "id": "26091888",
```

```
   "last_name": "Eliot",
   "link": "https://www.facebook.com/app_scoped_user_id/26091888/",
   "locale": "en_US",
   "name": "Tom Eliot",
   "timezone": 0,
   "updated_time": "2014-06-10T20:38:29+0000",
   "verified": true
}
```


Note

External identity provider APIs are subject to change. See the external identity provider's documentation for information.

Managing Resources

A resource is a collection, attribute, or thing that represents an organizational data asset. It is a multi-valued set of URNs (Unique Resource Names) that identifies the data being requested. Each URN can represent a Resource, a Resource attribute, or a sub-attribute.

For example, a Resource object might represent:

- an entire Resource `urn:unboundid:resource:user`
- a Resource attribute `urn:unboundid:resource:identity:User.address`
- or a sub-attribute `urn:unboundid:resource:identity:User.usernamepostalCode`

The resources listed on the Resources page represent the default SCIM and UserInfo schemas configured for the backend data stores.

To Create a New Resource

1. Click **Data Classification** and select **Resources**.
2. On the Resources page, click **New Resource**.
3. On the **General** tab, enter a name to identity the resource.
4. Enter an optional description for the resource.
5. In the **URN** box, enter the URN (unique resource name) for the resource. The format is `namespace:collection:attribute`.
6. Enter an optional trust level for the resource. An assigned trust level may affect a requesting application's access to it. If an application is considered less trusted than a resource, the application will not be allowed to access to that resource.
7. Enter an optional tag for the resource. For example, if a resource is tagged as HIPAA, only applications registered with the HIPAA tag can access it.
8. Click the **Aliases** tab.
9. Enter an alternate name for the resource.
10. Click the **Resource Groups** tab.

11. Select the group in the list to associate with the resource.
12. Click **Save**.

To Edit a Resource

1. Click **Data Classification** and select **Resources**.
2. On the Resources page, click the **Edit** button for the resource to modify.
3. On the **General** tab, change the resource name or description.
4. In the **URN** box, edit the URN (unique resource name) for the resource. The format is `namespace:collection:attribute`.
5. Enter or change the trust level for the resource.
6. Enter or change the tag for the resource.
7. Click the **Aliases** tab.
8. Enter an alternate name for the resource.
9. Click the **Resource Groups** tab.
10. Select the group in the list to associate with the resource.
11. Click **Save**.

To Delete a Resource

1. Click **Data Classification** and select **Resources**.
2. On the Resources page, click the drop-down arrow button next to the **Edit** button for the resource to delete.
3. Click **Delete**.
4. Confirm the action.

Managing Resource Groups

Resource Groups can be created for a collection of resources that do not necessarily fall within a single portion of the URN (Unique Resource Name) hierarchy. For example, request to access a resource `urn:unboundid:resource:address` would return all of its sub-attributes, such as `street_address`, `region`, `country`, and `postal_code`. Instead, a resource group can be created to include only certain sub-attributes, such as `street_address` and `postal_code`, not all that fall under the attribute hierarchy.

In order for access to a resource group to be granted to a client application, the application must request the group ID and an end user must consent to the group as an entity. If the group ID is not specified, an end user must consent to all resources contained in the group.

To Create a Resource Group

1. Click **Data Classification** and select **Resource Groups**.
2. On the Resource Groups page, click **New Resource Group**.
3. On the **General** tab, enter a name to identity the resource group.
4. Enter an optional description for the resource group.
5. In the **URN** box, enter the URN for the resource group. The format is `namespace:collection:attribute`.
6. Click the **Resources** tab.
7. Select the resources to add to the group.
8. Click **Save**.

To Edit a Resource Group

1. Click **Data Classification** and select **Resource Groups**.
2. On the Resource Groups page, click the **Edit** button for the resource group to modify.
3. On the **General** tab, change the name or description for the resource group.
4. In the **URN** box, enter the URN for the resource group. The format is `namespace:collection:attribute`.
5. Click the **Resources** tab.
6. Select the resources to add to the group.
7. Click **Save**.

To Delete a Resource Group

1. Click **Data Classification** and select **Resource Groups**.
2. On the Resource Groups page, click the drop-down arrow next to the **Edit** button for the resource group to delete.
3. Select **Delete**.
4. Confirm the action.

Managing Scopes

A scope indicates which data are being requested with an OAuth 2.0 authorization request. Typically, one or more scopes are submitted with each request. Scopes are tied to the resources that can be requested, the actions that can be taken on that resource, and the optional purpose for the actions.

Note

If a scope's action, purpose, or resource is updated, all tokens containing that scope will be revoked.

To Create a New Scope

1. Click **Data Classification** and select **Scopes**.
2. On the Scopes page, click **New Scope**.
3. On the **General** tab, enter a name for this scope. This name will be displayed to end-users when they are asked to provide consent.
4. In the **OAuth Name** box, enter the OAuth name for this scope. This is the name that client applications need to request the scope from the Identity Broker.
5. Enter an optional description for the scope.
6. To have a message display on an authorization approval page, enter text in the **Approval Page Summary** box.
7. Select the type of action to apply to this scope from the **Action** drop-down list.
8. Select the purpose for this scope from the **Purpose** drop-down list.
9. If the scope requires approval from the resource owner when requested, check the **Requires approval when requested** box.
10. Click the **Resources** tab.
11. Select the resources to add to the scope.
12. Click the **Resource Groups** tab.
13. Select the groups to apply to the scope.
14. Click **Save**.

To Edit a Scope

Any change to an existing scope's action or purpose will invalidate all tokens containing that scope. Before making changes to scopes, make sure that dependent tokens are no longer needed.

1. Click **Data Classification** and select **Scopes**.
2. On the Scopes page, click the **Edit** button for the scope to modify.
3. On the **General** tab, change the scope name. This name will be displayed to end-users.
4. In the **OAuth Name** box, enter the OAuth name for this scope.
5. Enter or change an optional description for the scope.
6. To have a message display on an authorization approval page, enter text in the **Approval Page Summary** box.

7. Select the type of action to apply to this scope from the **Action** drop-down list.
8. Select the purpose for this scope from the **Purpose** drop-down list.
9. If the scope requires approval from the resource owner when requested, check the **Requires approval when requested** box.
10. Click the **Resources** tab.
11. Select the resources to apply to the scope.
12. Click the **Resource Groups** tab.
13. Select the groups to apply to the scope.
14. Click **Save**.

To Delete a Scope

1. Click **Data Classification** and select **Scopes**.
2. On the Scopes page, click the drop-down arrow next to the **Edit** button for the scope to delete.
3. Click **Delete**.
4. Confirm the action.

Managing Data Views

Data Views provide a unified view of resources between the Identity Broker and one or more underlying data stores. Each Data View relies on a single Data View Schema, which must have the same name.

A Data View relies on a Data View Schema, which is imported into the Identity Broker Console. The Data View Schema is a JSON-formatted SCIM schema that is used to define what attributes can be retrieved from a backend data store. The Identity Broker provides a sample SCIM schema that can be used as a template in `<server-root>/resource/example-starter-schema`.

Each Data View represents one resource type, such as "user" or "account" and the schema defines the attributes of that resource. A `DataViewSchema` object is a Broker Store object, which is a container for the SCIM Resource Schema for that particular Data View.

Note

When mapping attributes, data store attributes and Data View 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.

See Using Data View Attributes in Policy for details about policy evaluation.

Simple Multivalued Attribute Mapping

If the Data View schema contains a multivalued attribute that has a "value" sub-attribute, the Identity Broker will consider this a "simple" multivalued attribute (a collection of simple attributes). In this case, only the "value" sub-attribute is mappable to a Store Adapter attribute. The Store Adapter attribute should be multivalued.

An example of this attribute type, from the SCIM 1.1 Core Schema specification, is the User `entitlements` attribute.

Complex Attribute Mapping

If the Data View schema contains a multivalued complex attribute that does not have a "type" sub-attribute (does not specify canonical types) and does not have a "value" sub-attribute, the Identity Broker can map that attribute to a multivalued string attribute in a Store Adapter. When the Identity Broker writes to the Store Adapter, each complex value of the Data View attribute is encoded as a JSON string. When reading from the Store Adapter, the JSON string is converted back into a complex structure.

All other complex attributes, including single-valued, simple multivalued attributes, and canonically-typed multivalued attributes, must be mapped at the sub-attribute level.

To Create a New Data View

Creating a Data View requires knowledge of the data stores and store adapters that have been implemented to support the identity infrastructure.

It also requires the manual creation of a Data View Schema that defines the attributes that can be mapped to resources in the backend user stores. This provides a unified view of a user's data across multiple backend user stores.

1. Click **Data Classification** and select **Data Views**.
2. On the Data Views page, click **New Data View**.
3. On the **General** tab, choose a Data View Schema to import into the Identity Broker. This file defines the attributes that will be mapped to resources in the back-end store adapters and must be in JSON format. See the SCIM Core Schema 1.1 Specification for information about formatting this file. A sample SCIM schema is located in `<server-root>/resource/example-starter-schema`.
4. Enter a name for this Data View. If not specified, the name will be populated from the Data View Schema file.
5. Enter an optional description for the Data View.
6. Enter the Schema URI (unique resource identifier). If not specified, the URI will be populated from the Data View Schema file.
7. Enter the Data View's endpoint HTTP address, which will be relative to the `/scim` base URL.

8. Enter an optional look-through limit for the maximum number of resources that the Data View should scan when processing a search request. This prevents a client from taking too many of the server's resources for a single search.
9. Click the **Enabled** check box to make the Data View's contents available for Identity Broker processing.
10. On the **Store Adapters** tab, choose one or more store adapters on which this Data View will rely for attribute mapping. Store adapters can be listed in the order in which attribute searches should take place.
11. Click **Save**.

To Edit a Data View

1. Click **Data Classification** and select **Data Views**.
2. On the Scopes page, click the **Edit** button for the Data View to modify.
3. On the **General** tab, choose a Data View Schema to import into the Identity Broker.
4. Change the name for this Data View. If not specified, the name will be populated from the Data View Schema file.
5. Enter or change the optional description for the Data View.
6. Change the Schema URI (unique resource identifier). If not specified, the URI will be populated from the Data View Schema file.
7. Change the Data View's endpoint HTTP address, which will be relative to the `/scim` base URL.
8. Enter or change an optional look-through limit for the maximum number of resources that the Data View should scan when processing a search request. This prevents a client from taking too many of the server's resources for a single search.
9. Click the **Enabled** check box to make the Data View's contents available for Identity Broker processing.
10. On the **Store Adapters** tab, choose one or more store adapters on which this Data View will rely for attribute mapping. Store adapters can be listed in the order in which attribute searches should take place.
11. On the **Attributes** tab, view the attributes that are populated by the schema file.
12. Click **Save**.

To Edit Store Adapter Mappings

1. Click **Data Classification** and select **Data Views**.
2. Click the drop-down arrow next to the **Edit** button for the Data View, and click **Edit Store Adapter Mappings**.
3. On the Edit Data View dialog, review the Data View Schema on the left and the related Storage Adapter Mapping on the right.
4. Click any of the attributes to be mapped between the Data View Schema and the storage adapter, or click **Edit All** next to the store adapter to list and edit all attributes.
5. For each Data View attribute, select the attribute to map from the drop-down list. If multiple store adapters are associated with this Data View, each will present a list of attributes.
6. Select the actions that can be performed on an attribute.
    - **Readable** – The Data View can read this attribute.
    - **Authoritative** – If there are multiple attributes of this type (from multiple data stores), one must be marked Authoritative.
    - **Writable** – The Data View can write to this attribute.
    - **Indexed** – This specifies whether the attribute is efficiently searchable in the underlying data store. Indexed data store attributes determine what attributes (from the Data View Schema) can be used in a SCIM filter when performing a query. If an attribute is not indexed in the data store, it should not be marked as indexed here.
7. Click **Done Editing**.
8. Click **Save** on the Edit Data View Store Adapter Mappings dialog.

To Export a Data View Schema

1. Click **Data Classification** and select click **Data View Schemas**.
2. On the Data View Schemas page, click the drop-down arrow next to the **Edit** button in a row.
3. Select **Export JSON**.
4. Open or save the file to an location.

To Delete a Data View

1. Click **Data Classification** and select **Data Views**.
2. On the Data Views page, click the drop-down arrow next to the **Edit** button for the Data View to delete.

3. Click **Delete**.
4. Confirm the action.

Managing External Identity Providers

Configuring External Identity Providers enables the Identity Broker to authenticate a user with a registered application. If the user does not already exist in any of the backend user stores, an account can be created with the username and other optional attributes.

The rules that determine how and when an identity can be created are defined in the User Create and Update Policy. See Managing Policies for more information.

The Identity Broker supports three provider types. Before configuring an external identity provider, the following is needed:

For Google

- Register the Identity Broker with Google to obtain a client ID and secret. See https://developers.google.com/accounts/docs/OAuth2 for details.
- Determine the identity provider's scopes that will be used for authorization. The default for Google is provided.
- Determine the Identity Broker Data Views that will be used to map a user's Google profile attributes to the Identity Broker backend user store.

For Facebook

- Register the Identity Broker with Facebook to obtain a client ID and secret. See https://developers.facebook.com/docs/web/tutorials/scrumptious/register-facebook-application/ for details.
- Determine the Identity Broker Data Views that will be used to map a user's Facebook profile attributes to the Identity Broker backend user store.
- Determine the Facebook permissions for which a user can grant access.

For an OpenID Connect Provider

- Register the Identity Broker with the provider to obtain a client ID and secret.
- Determine the Identity Broker Data Views that will be used to map a user's profile attributes to the Identity Broker backend user store.
- Determine the identity provider's scopes that will be used for authorization.
- Determine the authentication method for the client to use.
- Obtain the URLs of the provider's issuer identifier, the authorization endpoint, the token endpoint, and the userinfo endpoint.

- If using a custom OpenID Connect schema, create (or obtain) a JSON-formatted schema file to import. This enables the Identity Broker to use the provider's claims mapping for mapping attributes to the backend user store.
- The Identity Broker will use TLS server validation to authenticate OpenID Connect providers instead of validating ID token signatures when the token endpoint is secure (HTTPS). Only ID tokens signed with MAC-based algorithms such as HS256, HS384, or HS512 are supported when validating ID tokens over a non-secure connection.

To Create a New Identity Provider

1. Click **Data Classification** and select **External Identity Providers**.
2. On the External Identity Providers page, click **New External Identity Provider**.
3. On the **General** tab, enter a name for this provider.
4. Enter an optional description.
5. Select the identity provider type from the drop-down list. Supported providers are Facebook, Google, or OpenID Connect.
6. The Facebook and Google icons are available and will be added to the Identity Broker login page if configured. To add a custom icon to the login page, enter the location URI.
7. Select one or more Data Views to associate with the identity provider's attribute mapping.
8. On the **Connection** tab, supply the connection information for the selected identity provider. The options are specific to each provider type. The Identity Broker application must be registered with the provider.

For Facebook

1. Enter the **Application ID** that was given to the Identity Broker when it was registered with Facebook.
2. Enter the **Application Secret** that was given to the Identity Broker when it was registered with Facebook.
3. Select the **Data to Request** (Facebook permissions) from the drop-down list. These are the Facebook scopes that can be requested from a registered Identity Broker client application.

For Google

1. Enter the **Client ID** that was given to the Identity Broker when it was registered with Google.

2. Enter the **Client Secret** that was given to the Identity Broker when it was registered with Google.
3. Select the **Data to Request** (Google scopes) from the drop-down list. These are the Google scopes that can be requested from a registered Identity Broker client application.

For OpenID Connect

1. Enter the **Client ID** that was given to the Identity Broker when it was registered with the identity provider.
2. Enter the **Client Secret** that was given to the Identity Broker when it was registered with the identity provider.
3. Select the **Data to Request** (scopes) from the drop-down list. These are the identity provider scopes that can be requested from a registered Identity Broker client application.
4. Choose the authentication method to use when the Identity Broker client application connects to the identity provider's token endpoint.
5. Enter the URL that the identity provider recognizes as its issuer identifier.
6. Enter the URL for the identity provider's OAuth 2.0 authorization endpoint.
7. Enter the URL for the identity provider's OAuth 2.0 token endpoint.
8. Enter the URL for the identity provider's OAuth 2.0 userInfo endpoint.

To Edit an Identity Provider

1. Click **Data Classification** and select **External Identity Providers**.
2. On the External Identity Providers page, click the **Edit** button for the provider to modify.
3. On the **General** tab, change the name for this provider.
4. Enter or change an optional description.
5. The Facebook and Google icons are available and will be added to the Identity Broker login page if configured. To add a custom icon to the login page, enter the location URI.
6. Select one or more Data Views to associate with the identity provider's attribute mapping.
7. On the **Connection** tab, supply the connection information for the selected identity provider. The options are specific to each provider type. The Identity Broker application must be registered with the provider.

- For Facebook
- For Google

- For OpenID Connect

To Edit Identity Provider Mappings

1. Click **Data Classification** and select **External Identity Providers**.
2. On the External Identity Providers page, click the drop-down arrow next to the **Edit** button for the provider.
3. Click **Edit Data View Mappings**.
4. The attributes that can be requested for this identity provider are listed on the left. Attributes are based on the scopes or permissions that were configured when the provider was added. The Data Views that were configured for this provider are listed on the right.
5. To change the permissions that were configured for this provider, click **Select Permissions**. Permissions determine the attributes that are listed for the provider.
6. To add a custom claim to the provider's list, click **Add Custom Claim**. The custom claim can be mapped to any of the Data View attributes.

Note

In order for custom claims with multiple levels of nested attributes to be mapped, each nested attribute must be added as individual claims. This only applies if user attributes will be updated based on a claim retrieved from an OpenID Connect identity provider. Claims with top level JSON arrays are supported and represented as multi-valued SCIM attributes. However, claims with nested JSON structures containing arrays are not supported and are not mappable.

7. If more than one Data View was configured, click **Edit All** at the top of the list to display all Data Views.
8. Click **Define Mapping** for each attribute that should be mapped to a Data View.
9. Select the Data View attribute to map from the drop-down list.
10. If the value for the Data View attribute should overwrite the identity provider attribute value, select **Always Overwrite** or **Overwrite Only if Value is Missing**. If the Data View value should not be overwritten, select **Never Overwrite**.

Note

Mappings for attirbutes used for login, such as username or id, should have the update option set to "Never Overwrite."

11. Click **Done Editing** to save settings.
12. Click **Save** when finished mapping attributes.

To Delete an Identity Provider

1. Click **Data Classification** and select **External Identity Providers**.
2. On the External Identity Providers page, click the drop-down arrow next to the **Edit** button for the provider to delete.
3. Click **Delete**.
4. Confirm the action.

Managing UserInfo Mappings

A UserInfo endpoint is an OAuth 2.0 protected resource that returns information about an authenticated end user. UserInfo Mapping enables mapping Data View Schema 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.

Note

A UserInfo map should be created for each Data View.

UserInfo Claims and Scopes

For a client application to successfully retrieve an OpenID Connect claim from the UserInfo endpoint, it must request and get consent to use a corresponding scope. However, claims correspond to specific resources, which are contained within scopes. Make sure that configured scopes contain the resources that client applications will request from the UserInfo endpoint. Make sure that any changes to resource mapping are also made in the scope configuration.

For example, if the `address` UserInfo claim is configured to return work addresses, the `address` scope must be changed as well. By default, the `address` scope is mapped to `urn:scim:schemas:core:1.0:addresses.preferred`. It should be updated to `urn:scim:schemas:core:1.0:addresses.work`. If the UserInfo mapping and the scope configuration do not match, a client applications that requests the `address` scope will have no value returned for the UserInfo endpoint's `address` claim.

Complex Attribute Mapping

If an attribute is complex (such as `urn:scim:schemas:core:1.0:name`, or `urn:scim:schemas:core:1.0:addresses.preferred`), 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 Data View attribute that corresponds to the SCIM core attribute. Complex attribute mapping can only be done through the `broker-admin` tool, except for the standard `address` claim. The Identity Broker Console automatically displays the sub-attributes for the `address` claim in the Edit dialog.

To Create a New UserInfo Mapping

1. Click **Data Classification** and select **User Info Mappings**.
2. On the User Info Mappings page, click **New User Info Mapping**.
3. On the **General** tab, enter a name for the new mapping and optional description.
4. Select a Data View Schema from the drop-down list.
5. On the **OIDC Claim Mapping** tab, map any of the schema attributes to an OIDC Claim.
6. Click **Save**.

To Edit a UserInfo Map

1. Click **Data Classification** and select **User Info Mappings**.
2. On the User Info Mappings page, click **Edit** for the item to modify.
3. On the **General** tab, change the name for the mapping and optional description.
4. Select or change a Data View Schema from the drop-down list.
5. On the **OIDC Claim Mapping** tab, map any of the schema attributes to an OIDC Claim.
6. Click **Save**.

To Delete a UserInfo Map

1. Click **Data Classification** and select **User Info Mappings**.
2. On the User Info Mappings page, click the drop-down arrow next to the **Edit** button for the item to delete.
3. Select **Delete**.
4. Confirm the action.

Chapter 4: Categories

Categories organize and identify applications and resources using **Tags** and **Trust levels**. Both Tags and Trust levels need policies enabled to have an affect on authorization decisions. The Governance Tag Policy and the Trust Level Policy are installed as default policies during the configuration of the Identity Broker. These policies can be customized or new policies can be written based on configured Tags and Trust levels.

If an application is given a governance tag or trust level assignment, the assignment must correspond to any tags or trusts assigned to the resources that the application will request. For example, if an application is considered less trusted than a resource, the default Trust Level policy will not allow the application to access that resource. If a resource is tagged as HIPAA, the default Governance Tag policy will only allow applications with the HIPAA tag to access it.

This section explains how to manage data category components in the Identity Broker Console application. In this section, the following tasks are performed:

Managing Tags

Tags are optional labels assigned to applications or resources for organization and management. Tags can be created for identifying regulatory or business standards, such as HIPAA (Health Insurance portability and Accountability Act) or GLBA (Gramm–Leach–Bliley Act), or Tags can also be used in policies, so that a single policy can be used for applications or resources based on the tag.

The Governance Tag Policy is similar to Consent policy in that it accounts for the hierarchical nature of resources. An application is granted access to a resource if it possess all governance tags that are associated with the resource.

Note

If a Resource has no Tags assigned to it, it will inherit the Tags of its first ancestor that does.

To Create a New Tag

1. Click **Categories** and select **Tags**.
2. On the Tags page, click **New Tag**.
3. On the **General** tab, enter a name to identity the tag.
4. Enter an optional description for the tag.
5. Enter an optional URL for reference information about the tag. This is generally for background information related to the purpose of the tag.
6. Click the **Applications** tab.
7. Select the applications to which this tag will apply.
8. Click the **Resource** tab.
9. Select the resources to which this tag will apply.
10. Click **Save**.

To Edit a Tag

1. Click **Categories** and select **Tags**.
2. On the Tags page, click the **Edit** button for the tag to modify.
3. On the **General** tab, change the tag name.
4. Enter or change an optional description for the tag.
5. Enter or change an optional URL for reference information about the tag.
6. Click the **Applications** tab.
7. Select the applications to which this tag will apply.
8. Click the **Resource** tab.

9. Select the resources to which this tag will apply.
10. Click **Save**.

To Delete a Tag

1. Click **Categories** and select **Tags**.
2. On the Tags page, click the drop-down arrow next to the **Edit** button for the tag to delete.
3. Click **Delete**.
4. Confirm the action.

Managing Trust Levels

Each optional trust level has a single numeric value that is assigned to an application or a resource. For example, an internally developed application can be assigned a higher trust level than a third-party application. A resource representing a social security number would be assigned a higher trust level than a favorite color resource.

Unlike Governance Tags, only one trust level can be associated with a resource or application. If trust level *X* is associated with a resource, then a requesting application must have a trust level greater than or equal to *X* to be granted access to the resource.

The Trust Level Policy ensures that if any resource has a trust level, the application requesting access to that resource must have trust level greater or equal to the highest trust level of the resource in the request. For example, if you define a "Medium Trust Level" and give it a value, then assign that trust level to a Personal Phone Number resource, any application requesting access to the resource must have the same trust level or greater.

Note

If a Resource has no Tags (or Trust Level) assigned to it, it will inherit the Tags (or Trust Level) of its first ancestor that does.

To Create a New Trust Level

1. Click the **Categories** and select **Trust Levels**.
2. On the Trust Levels page, click **New Trust Level**.
3. In the **Name** box, enter a name to identify the trust level.
4. Enter an optional description for the trust level.
5. Enter a numeric value in the **Value** box, or move the slider from lowest to highest to set a value for the trust level.
6. Click **Save**.

To Edit a Trust Level

1. Click the **Categories** and select **Trust Levels**.
2. On the Trust Levels page, click the **Edit** button for the trust level to modify.
3. Change the name of the trust level.
4. Enter or change an optional description for the trust level.
5. Change the numeric value in the **Value** box, or move the slider from lowest to highest to set a value for the trust level.
6. Click **Save**.

To Create a New Trust Level more than the Selected Trust Level

1. Click **Categories** and select **Trust Levels**.
2. On the Trust Levels page, click the drop-down arrow next to the **Edit** button for a trust level.
3. Select **New Trust Level more than <listed trust level>**.
4. Enter a name to identify the trust level in the **Name** box.
5. Enter an optional description for the trust level.
6. The default value of the new trust level will be incremented by one for the selected trust level. Enter a higher value, or move the slider from lowest to highest to set a value.
7. Click **Save**.

To Create a New Trust Level less than Selected Trust Level

1. Click **Categories** and select **Trust Levels**.
2. On the Trust Levels page, click the drop-down arrow next to the **Edit** button for a trust level.
3. Select **New Trust Level more than <listed trust level>**.
4. Enter a name to identify the trust level in the **Name** box.
5. Enter an optional description for the trust level.
6. The default value of the new trust level will be incremented by one less for the selected trust level. Enter a lower value, or move the slider from the highest to lowest to set a value.
7. Click **Save**.

To Delete a Trust Level

1. Click the **Categories** and select **Trust Levels**.
2. On the Trust Levels page, click the drop-down arrow next to the **Edit** button for the trust level to delete.
3. Click **Delete**.
4. Confirm the action.

Chapter 5: Policies

Policies are the rules that determine what data is shared with data requestors or client applications, and for what action or purpose.

Policies are specified with a syntax defined in the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0* (http://docs.oasis-open.org), and can contain targets, rules, conditions, and a rule combining algorithm.

Sample policies can be installed with the initial Identity Broker configuration. A sample template (`IPAddressTemplate.xml`) is available in the `/resources` directory of the Identity Broker distribution.

This section explains how to manage policy components in the Identity Broker Console application. In this section, the following tasks are performed:

Policy Engine Request Context

The input to any XACML authorization policy is the request context. The request context contains attributes directly passed by a (PDP) client when making an authorization request to the policy engine. It can be supplemented with additional attributes that are retrieved from the "environment" by the PDP's Context Handler. In order to make a policy decision, policies can reference any attribute from the request context including those that come from the larger environment. In the case of the Identity Broker, this larger environment includes attribute values that may be retrieved from the Identity Broker's Data Views, as well as the value of pertinent objects from the Broker Store.

Writing and/or customizing policies in the Identity Broker requires knowledge not only of the XACML markup language but also of the specific ways in which the Identity Broker utilizes policy, such as:

- Which of the standard XACML request attributes are supported by the Identity Broker?
- What custom XACML request attributes are defined by the Identity Broker?
- What custom XACML functions are defined by the Identity Broker?
- Under what conditions does the Identity Broker generate a policy request?
- What are the details of each type of policy request generated by the Identity Broker?
- Which optional features defined by XACML 3.0 are not supported by the Identity Broker?
- How are Data View attributes used when making policy decisions?

After policies are written they should be tested to make sure the information that is accessed or denied is as expected. See Managing Policy Tests and Testing for more information.

How Policy Affects the Data Returned to an Application

The policies defined by the Identity Broker administrator will determine the resources that are returned to the client application. For example, if the client application requests the OpenID Connect scope `profile`, the policies defined for the Identity Broker may restrict access to sensitive attributes such as `birthDate` and `userName`, but return other attributes within that scope.

This Attribute-Based Access Control (ABAC) model delivers partial results instead of denying access to all attributes in the scope. If an application request to the Identity Broker is delivering partial results, it may be due to policy settings.

About Data Access Requests

The Identity Broker's policy engine governs the conditions by which an application can access resources. Creating policies requires understanding the structure of a data access request. If default policies were installed, the Consent Policy grants access to data requests based on consent from the resource owner (usually an end user).

A request consists of the following parameters:

Subject – Identifies the application requesting access to specified resources.

Action – Identifies the operation that the application would like to perform on the specified resources, such as "read."

Consent Owner – Identifies the owner who has the authority to grant permission to the subject for action on the specified resources.

Purpose – Identifies the reason for the subject's request to access the specified resources. This parameter is optional.

Resource – Identifies one or more sets of URNs (Uniform Resource Names) that identify the data being requested. Each URN can represent a resource attribute or a resource group. The representation of these is hierarchical. This hierarchy is important for policy evaluation. A top-level resource collection is considered the ancestor, and any lower level resources or attributes are considered descendants. For example,

- `urn:scim:schemas:core:1.0:name`, represents the components of a user's name.
- `urn:scim:schemas:core:1.0:name.familyName`, represents a resource as a sub-attribute of the complex name attribute.

Resource Groups, like resources, are also identified with a URN. A resource group represents a set of resources that are not in a hierarchy. The advantage of creating resource groups is that a request can specify the group and not need to specify all of the attributes in a resource hierarchy.

See Data Classification for more information about resources and resource groups.

About Policy Evaluation

For a policy to be evaluated against an authorization 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 Identity Broker 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 applicable policies are to be combined to form an ultimate decision by the Identity Broker. By default, the combining algorithm for Identity Broker policies is `deny-overrides`. This can be changed with the `dsconfig` tool. See the *UnboundID Identity Broker Installation Guide* for details.

Accessing Resources by Consent

A requested resource can be either a resource or a resource group. Access is granted to a resource if one of the following is true:

- A consent object contains an exact match on the resource ID.
- A consent object contains an ancestor of the resource ID.
- A consent object contains a resource group, of which the resource is a member.
- A consent object contains a resource group, of which an ancestor of the resource is a member.
- Consent has been granted to all descendant resources of the resource.

Consent is granted to a resource group if one of the following is true:

- A consent object contains an exact match on the resource group ID.
- Consent has been granted to all members of the resource group.

Policy Authorization Scenarios

Policies are evaluated by the Identity Broker in response to the following requests made by client applications:

- An authorization/token request to the OAuth 2.0 endpoint.
- A request to the UserInfo endpoint.
- All SCIM requests:
    - Search request
    - Get request
    - Update request
    - Create request
    - Delete request
- Self registration request.

- All requests to the Metadata API.
- A XACML request to the PDP endpoint.

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

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 Identity Broker for evaluation. The request is passed directly to the policy engine. The request can contain any standard XACML attributes, Identity Broker custom attributes, or other attributes that may be required by custom policies. This endpoint requires that the client authenticate using bearer token authentication, and that the token must have the `urn:unboundid:scope:invoke_pdp` scope.

Policies and Request Processing Per Endpoint

Requests from a client application are evaluated by the policy rules configured for the Identity 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.

Note

The `Any` purpose, if added to a scope, will match any purpose value. If a scope is created without an explicit purpose, `Any` will be assigned to it. This is important for OAuth 2.0 and UserInfo endpoint processing.

Requests Through the OAuth 2.0 Endpoint

Requests coming through the OAuth 2.0 endpoint are given an access token if the scopes specified are allowed by configured policies. Only the scope is granted or denied, not the resources contained within the scope. The token returned may not be valid for all the scopes that were included in the original request. The client application will receive a list of approved scopes with the access token. If all scopes are denied, then no access token is issued.

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.

Requests Through the UserInfo Endpoint

A request to the UserInfo endpoint has no arguments other than the access token itself. A UserInfo request is authorized with a single XACML request. The data returned is limited to the resources included in the scopes that were granted in the token.

Requests Through the SCIM Endpoint

A request to the SCIM endpoint includes the token and arguments that describe which attributes the requestor would like to retrieve. The request can contain attributes that are not

granted by the token. Policies are checked again to make sure nothing is returned that is not allowed.

The following actions are submitted in the generated XACML request depending on the SCIM operation being performed.

Action Performed Based on XACML Request

SCIM Operation Type	Action in XACML request
POST	Create
GET	Read
PATCH or PUT	Update
DELETE	Delete

Example Request Flow

For example, if an application requested access to Scope A and Scope B, the following would be considered:

- Scope A contains resources 1, 2, and 3.
- Scope B contains resources 4 and 5.
- Policy evaluation determines that access to resources 1, 2, 4, and 5 can be granted. Resource 3 is denied.
- Because one of the resources in Scope A is denied, the scope is not included in the access token sent back to the client application. The token contains a grant for Scope B.
- If the client application sends a request with the access token to the UserInfo endpoint, only the resources in Scope B are returned.
- If the client application sends a request for resources 1, 2, 3, 4, and 5 (with the access token) to the SCIM endpoint, Policy is reevaluated, and only resources 1, 2, 4, and 5 are returned.

OAuth 2.0 Endpoint Policy Evaluation

The OAuth 2.0 endpoint relies on the policy engine to determine whether an access token or authorization code should be granted to a requesting client. An independent XACML request is evaluated for each scope requested by the client. The token that is issued to the client may be valid for only a subset of the scopes originally requested.

The attributes included in the XACML request will vary depending upon the OAuth 2.0 grant type being requested. See the *UnboundID Identity Broker Application Developer Guide* for details about OAuth 2.0 grant types.

Authorization Code and Implicit Grant Types

Because of the interactive nature of these two OAuth 2.0 flows, the OAuth 2.0 endpoint splits policy checking into two phases. The first phase checks whether the token request would be allowed by all installed policies except for consent policy. If the result of this first phase is DENY then the second phase is not executed.

The second phase checks whether the end user's consent is required before the requested scope can be granted. If so the flow proceeds to prompt the user for consent. If the second phase indicates that the user's consent is not required (either by rule or because they have already consented), then the OAuth 2.0 endpoint issues the requested token or authorization code.

The phase one XACML request contains the attributes below. It is executed once for each scope in the token request. Note that resource owner is not included in the request, which results in the consent policy (which is based upon resource ownership) to not be applied.

XACML Attribute	Attribute Value
`actor-id`	SCIM Id of the currently authenticated user.
`subject-id`	Application name, obtained from the OAuth request's client ID parameter.
`action-id`	Action name obtained from the scope definition.
`purpose-id`	Purpose name obtained from the scope definition.
`resource-id`	Bag of resource URNs, obtained from the scope definition.

The phase two XACML request is sent to the OAuth Consent Evaluation policy sandbox rather than to the global policy engine. This results in only consent policy being applied to the request. This request contains the attributes specified in the table below.

XACML Attribute	Attribute Value
`owner-id`	SCIM ID of the currently authenticated user (for OAuth requests, owner ID is always the same as the actor ID).
`actor-id`	SCIM ID of the currently authenticated user.
`subject-id`	Application name, obtained from the OAuth request's client ID parameter.
`action-id`	Action name obtained from the scope definition.
`purpose-id`	Purpose name obtained from the scope definition.
`resource-id`	Bag of resource URNs, obtained from the scope definition.

The OAuth Consent Evaluation sandbox isolates consent checking from other policies. The contents of the sandbox may be modified in order to customize consent policy, however the sandbox itself cannot be deleted.

Client Credentials Grant Type

A client credentials OAuth request is a request by an application for access to its own resources. It does not require that a user currently be authenticated to the Identity Broker. Like all OAuth interactions, one policy evaluation is made for each scope requested. The attributes of the XACML request generated for this grant type are specified in the table below.

XACML Attribute	Attribute Value
`subject-id`	Application name.
`action-id`	Action name obtained from the scope definition.
`purpose-id`	Purpose name obtained from the scope definition.
`resource-id`	Bag of resource URNs, obtained from the scope definition.

Resource Owner Grant Type

The Resource Owner grant type does not require consent. In general, only trusted applications should be allowed to use this grant type. It evaluates policy independently for each scope contained in the request. Each XACML request is identical to that specified in phase one of the Authorization Code and Implicit Grant Types.

UserInfo Endpoint Policy Evaluation

A request to the UserInfo endpoint does not require any parameters other than an OAuth2.0 access token. The scopes represented by the token indicate what resources and attributes are being requested by the client application, and the token's owner identifies the resource owner. (Since a client credentials token has no owner, it cannot be used with the UserInfo endpoint.)

UserInfo is a read-only interface. Any scopes whose associated action is not `read` are discarded. The UserInfo endpoint also consults the Claims Map for the user's Data View and will only do policy checks on resources that are mapped through the Claims Map.

A single request to the UserInfo endpoint will result in several XACML policy evaluations since the access token can represent multiple scopes, and each scope can represent many resources. Each resource is evaluated independently by policy, and only those resources that are permitted by policy are returned as claims to the client application.

Each XACML request generated by UserInfo contains the following attributes:

XACML Attribute	Attribute Value
`owner-id`	SCIM ID of the access token owner.
`subject-id`	Name of the application associated with the access token.
`action-id`	Always set to "Read."
`purpose-id`	Purpose name obtained from a scope associated with the access token.
`resource-id`	A single resource URN obtained from the same scope.

SCIM Endpoint Policy Evaluation

Each request to the SCIM endpoint explicitly specifies what action is being requested and on what resources. As a REST interface, SCIM uses the HTTP method, query parameters, method body, and URI path to specify request parameters. Policy evaluations generated by the SCIM endpoint depend on these REST parameters, as well as the supplied OAuth 2.0 bearer token, which is used mainly for authentication.

All SCIM requests target a specific Data View. For all request types, the SCIM endpoint first consults the appropriate Data View mapping and will pare out any unmapped request attributes before it generates policy requests.

For example, a search targeted to `/scim/Users` is executed against the Users Data View. An update targeted to `/scim/ConsumerUsers/9f8a23-5f7ec932-55c4-347e-b757-ce74258ea9e6` is executed against a user with ID `9f8a23-5f7ec932-55c4-347e-b757-ce74258ea9e6` in the ConsumerUsers Data View.

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 Data View against which the search is to be conducted is derived from the URI path, such as `/scim/Users`.

After the SCIM endpoint executes the search against the Data View, it generates XACML requests for each record returned in the search results in order 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.

Note

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

Each XACML request contains the following attributes:

XACML Attribute	Attribute Value
`owner-id`	SCIM ID of the returned result record.
`actor-id`	SCIM ID of the OAuth 2.0 access token owner. This attribute will not be included in the request if the access token was obtained through a Client Credentials grant.
`subject-id`	Application name of the requesting application, retrieved from the OAuth access token.
`action-id`	Always "Read," since this is a search request.
`purpose-id`	Always "Any," since the SCIM standard does not include a purpose specification.
`resource-id`	A single Resource URN from the returned result record.

Any resources or individual resource attributes that are denied by policy are omitted from the search response.

SCIM Get Request

A SCIM request to obtain a single record is handled similarly to the search request, except that there is only a single result record. The previous table applies.

SCIM Update Request

A SCIM update request (HTTP PATCH) contains in the message body the attributes to be updated and/or deleted. Deleting an attribute from a record is considered an update action by the SCIM endpoint. The response to an update request contains the updated record. Using query attributes the SCIM client can request that only a subset of the updated record be returned in the response.

The SCIM endpoint issues two sets of policy evaluations in response to an update request. The first set determines which attributes the client is permitted to update. These XACML requests contain the following:

XACML Attribute	Attribute Value
`owner-id`	SCIM ID of the record to be updated.
`actor-id`	SCIM ID of the OAuth 2.0 access token owner. This attribute will not be included in the request if the access token was obtained through a Client Credentials grant.
`subject-id`	Application name of the requesting application, retrieved from the OAuth 2.0 access token.
`action-id`	Always “Update.”
`purpose-id`	Always “Any,” since the SCIM standard does not include a purpose specification.
`resource-id`	A single Resource URN obtained from the request’s message body.

An `update-operation` attribute (`urn:unboundid:names:1.0:update-operation`) is present in the request context when the value of `action-id` is `Update`. It is populated by the SCIM endpoint to provide information about the type of update being performed. Currently, only `delete` is supported, which is set when a request updates a record by deleting an attribute or deleting a value from a multivalued attribute. If the attribute is not present as part of an update request, a policy may assume that the update is either replacing or adding an attribute value.

Note

The policy engine has access to the resource URN, but not the proposed new value for the corresponding attribute. Therefore, policy can check whether the application is allowed to update the attribute, but cannot do data validation on the attribute value.

After the update is complete, a second set of policy requests is issued to determine which attributes of the updated record the client can receive in the response. These requests are formatted exactly as for a SCIM Get or Search request.

SCIM Create Request

Like an update request, a SCIM create request contains the attributes of the new record in the message body. The response to the request is the contents of the new record, which optionally can be pared by query parameters that specify which attributes the client wants to receive in the response.

Policy checks for SCIM create requests (HTTP POST) are different in that there is no existing resource owner. The owner is being created as a result of the request. Also, the entire set of attributes is evaluated by a single XACML request. Either the entire request is accepted or denied, there is never a partial success where some attributes are saved but not others. The create policy request therefore contains attributes as follows:

XACML Attribute	Attribute Value
`actor-id`	SCIM ID of the OAuth 2.0 access token owner. This attribute will not be included in the request if the access token was obtained through a Client Credentials grant.
`subject-id`	Application name of the requesting application, retrieved from the OAuth 2.0 access token.
`action-id`	Always “Create.”
`purpose-id`	Always “Any,” since the SCIM standard does not include a purpose specification.
`resource-id`	A list of all resource URNs specified in the request’s message body.

Note

The policy engine has access to the resource URN, but not the proposed new value for the corresponding attribute. Therefore, policy can check whether the application is allowed to update the attribute, but cannot do data validation on the attribute value.

SCIM Delete Request

A SCIM delete request is a request to delete a record from the underlying Data View. To determine whether the delete request should be permitted, the SCIM endpoint will invoke the policy engine with a XACML request that includes the following attributes:

XACML Attribute	Attribute Value
`owner-id`	SCIM ID of the record to be deleted.
`actor-id`	SCIM ID of the OAuth 2.0 access token owner. This attribute will not be included in the request if the access token was obtained through a Client Credentials grant.
`subject-id`	Application name of the requesting application, retrieved from the OAuth 2.0 access token.
`action-id`	Always “Delete.”
`purpose-id`	Always “Any,” since the SCIM standard does not include a purpose specification.
`resource-id`	A list of all top-level resource URNs defined by the Data View schema.

Self-Registration Policy Evaluation

Self-registration is an unauthenticated activity that allows a visitor to an application site to create an account. A request to the Identity Broker’s registration endpoint is a HTTP POST whose content must include the requesting application’s client ID, the name of the Data View in which to register the new user, and the new user’s attribute values. The registration endpoint constructs a XACML request from these arguments so that the policy engine can evaluate

whether the registration should be allowed. The XACML request is formatted with the following attributes:

XACML Attribute	Attribute Value
`subject-id`	Name of the requesting application.
`action-id`	Always “Create.”
`purpose-id`	Always “Registration.”
`resource-id`	A list of all resource URNs specified in the request’s message body.

Metadata API Policy Evaluation

Calls to the Metadata API require an OAuth 2.0 access token and are subject to approval by policy. The operations supported by the Metadata API include:

- creation, deletion, and retrieval of user consents
- retrieval of user consent history
- retrieval of user access history
- creation, deletion, and retrieval of user external identity provider links

The exact policy request generated by the Metadata endpoint will depend on which API is invoked, but in general will contain the following attributes:

XACML Attribute	Attribute Value
`owner-id`	SCIM ID of the user whose metadata is being accessed.
`actor-id`	SCIM ID of the OAuth 2.0 access token owner. This will always be present as a Client Credentials token is not allowed by the Metadata API.
`subject-id`	Application name of the requesting application, retrieved from the OAuth 2.0 access token.
`action-id`	Either “Read” or “Update,” depending on which Metadata API has been invoked. Creation or deletion of consents and identity provider links are considered updates to a user’s record, therefore the action will be “Update” for those methods.
`purpose-id`	Always “Any.”
`resource-id`	The resource URN(s) to which access is being requested. These resources are predefined by the Identity Broker and will always begin with `urn:unboundid:resources:broker_metadata:`. For a complete list of metadata resource URNs, see the *UnboundID Identity Broker Application Developer Guide*.

Policy Writing Guidelines

Policies are the rules or set of rules that determine whether resources can be accessed, by whom, under what conditions, and for what action. The basic outline for a policy looks like this:

```
Policy
   xmlns
   PolicyID
```

```
   RuleCombiningAlgID
   Description
   Target
   Rule
      RuleID
      Effect
      Description
      Target
      Condition
```

A policy must have:

- A unique identifier (PolicyID).
- A rule combining algorithm. This is a XACML attribute that defines the procedure for making an authorization decision given the results of a set of rules within a policy or for all policies in a policy set.
- A target, which determines the requests to which a policy applies.
- One or more rules, which includes a target, an effect, and a condition for making a request decision.

Policies can also contain Actors, Actions and Purposes. These settings are used to narrow access to particular applications or users for specific actions.

Policies can be edited or exported in XACML format and used as a template for new policies. Several policies are available after installation, if the option to install default policies was selected.

See the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0* (http://docs.oasis-open.org) for detailed information about policy components.

About Policy Templates

Policy templates are basically policies with parameterized attribute values. A policy template specifies one or more `<AttributeValue>` elements with `ParameterId` extensions containing string names. When a template is used to create a Policy, the process binds specific data to the parameters defined in the template. A policy template can be used to create any number of policies. A policy can refer to only one template. Policy templates are stored in the Broker Store as separate object types from policies.

Policy Template Parameters and Attributes

A policy template is useful for creating policies that are similar in structure, but maybe values change slightly or can be represented as a range. Two parts of a policy support parameterization: the `<Target>` specification and `<Rule>` specifications. In both cases, use the `ParameterID` extension in an `<AttributeValue>` element to declare variable parameters. The standard format for `<AttributeValue>` is:

```
<AttributeValue DataType=”http://www.w3.org/2001/XMLSchema#string”>
     SomeConstantValue</AttributeValue>
```

`DataType` can be any XACML data type. The element value is a string representation of that data type.

For policy templates only, the following extension attributes may be specified in an `<AttributeValue>`:

```
<AttributeValue ParameterId=”ParameterName”
    DataType=”http://www.w3.org/2001/XMLSchema#string”/
    Description=”Helpful description goes here”
    Validation=”Optional validation regex”/>
```

There is no element value. The `<AttributeValue>` references a parameter `“ParameterName”` of type `“string.”` If `<AttributeValue>` is contained within a `<Rule>`, `"ParameterName"` can be multi-valued. When a template becomes a policy, a set of one or more literal strings must be supplied for the parameter. If two values `alpha` and `beta` are specified, a policy created from the previous template `<AttributeValue>` would look like:

```
<AttributeValue DataType=”http://www.w3.org/2001/XMLSchema#string”>
    alpha</AttributeValue>
<AttributeValue DataType=”http://www.w3.org/2001/XMLSchema#string”>
    beta</AttributeValue>
```

Standard XACML Attribute Use

The following request attributes are specified by XACML 3.0. Unless otherwise specified, these are always available in the Identity 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 application that is submitting a policy request, as specified when the application is registered with the Identity 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 Identity Broker’s network environment. When available, the value is retrieved from the XFORWARDED_FOR header of the client’s HTTP

Attribute URN	Attribute Category	XACML Data Type	Description
			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. May be several of these per request.
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, as specified by a corresponding Action object in the Broker Store. For requests generated internally by the Identity Broker, this will be one of {“Read”, “Update”, “Create”, or “Delete”}.
urn:oasis:names:tc:xacml:2.0: action:purpose	urn:oasis:names:tc:xacml:3.0: attribute-category:action	string	Contains the purpose for which the authorization request is being made. Purposes are identified by the name given to Purpose objects contained in the Broker Store. This attribute is optional and may not be present for some authorization requests.
urn:oasis:names:tc:xacml:1.0: environment:current-time	urn:oasis:names:tc:xacml:3.0: attribute-category:environment	time	The time at which the Identity 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 Identity Broker began processing the current authorization request.

Custom XACML Attribute Use

The attributes in this section are defined by UnboundID for the Identity Broker. They will be added to the request context by the Identity Broker when applicable. Per the XACML specification, any attribute retrieved from the request context with an `AttributeDescriptor` element will be a bag of attribute values. Where an 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").

Custom XACML Attributes

Attribute URN	Attribute Category	Xacml Data Type	Description
urn:unboundid:names:1.0:httpQueryParameter	urn:oasis:names:tc:xacml:3.0:attribute-category:environment	string	Prefix that can be used by policies to reference any query parameter that was provided in the current HTTP request. For example, to reference the value of a query parameter named `channel`, the attribute URN is `urn:unboundid:names:1.0:httpQueryParameter:channel`.
urn:unboundid:names:1.0:httpHeader	urn:oasis:names:tc:xacml:3.0:attribute-category:environment	string	Prefix that can be used by policies to reference the value of a header value from the current HTTP request. For example, to reference the value of the Authorization header, the attribute URN is `urn:unboundid:names:1.0:httpHeader:Authorization`.
urn:unboundid:names:1.0:update-operation	urn:oasis:names:tc:xacml:3.0:attribute-category:action	string	This attribute is present in the request context only when the value of `action-id` is `Update`. It is populated by the SCIM endpoint to provide information about the type of update being performed. Currently, only `delete` is supported, which is set when a SCIM patch request updates a record by deleting an attribute or deleting a value from a multivalued attribute. If the attribute is not present as part of an update request, a policy may assume that the update is either replacing or adding an attribute value.
urn:unboundid:names:1.0:owner:owner-id	urn:unboundid:names:1.0:attribute-category:resource-owner	string	Contains the unique SCIM ID of the owner of the resources being requested. Multiple owner attributes can be included in the request and each will be evaluated separately. The policy engine will return a decision for each combination of owner and resources.
urn:unboundid:names:1.0:owner:dataview-name	urn:unboundid:names:1.0:attribute-category:resource-owner	string	If the request context contains an `owner-id` attribute, this attribute will contain the name of the Data View in which the owner's attributes are stored.
urn:unboundid:names:1.0:consent:captured-consent	urn:unboundid:names:1.0:attribute-category:resource-owner	consent (UnboundID data type extension)	If the request context contains an `owner-id` attribute, this attribute contains a XACML bag of consent records. The format of this attribute is proprietary and is not intended for use outside of the Identity Broker's `get_consent_status` function.
urn:unboundid:names:1.0:actor:actor-id	urn:unboundid:names:1.0:attribute-category:actor	string	Contains the unique SCIM ID of the currently authenticated user of the requesting application.
urn:unboundid:names:1.0:actor:dataview-name	urn:unboundid:names:1.0:attribute-category:actor	string	If the request context contains an `actor-id` attribute, then this attribute will contain the name of the Data View in which the actor's attributes are stored. (This is not available when applications are using the OAuth 2.0 client credentials flow.)
urn:unboundid:nam	urn:oasis:names:tc:xacml	string	Contains a XACML bag of Tag names associated with

Attribute URN	Attribute Category	Xacml Data Type	Description
es:1.0:subject:governance-tags	:1.0:attribute-category:access-subject		the subject (application).
urn:unboundid:names:1.0:resource:governance-tags	urn:oasis:names:tc:xacml:3.0:attribute-category:resource	string	Contains a XACML bag of Tag names associated with the resource being requested. If multiple resources are being requested, this attribute will contain the union of all Tags associated with the specified resources.
urn:unboundid:names:1.0:subject:trust-level	urn:oasis:names:tc:xacml:1.0:attribute-category:access-subject	integer	Contains the Trust Level associated with the subject (application).
urn:unboundid:names:1.0:resource:trust-level	urn:oasis:names:tc:xacml:3.0:attribute-category:resource	integer	Contains the Trust Level associated with the requested resource. If multiple resources are requested, this attribute will contain a bag of Trust Levels, one per specified resource.

Identity Broker Custom XACML Function

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

`urn:unboundid:names:1.0:function:get-consent-status`. Returns a XACML boolean value indicating whether a user has consented to the authorization request represented by the input arguments, which are:

- Application name (string): name of application making the request.
- Action name (string): name of action being requested, such as `read`.
- Purpose name (string): purpose name for the request.
- Consents (list of consent): raw consent records of the resource owner, retrieved from the Identity Broker's metadata store.
- Resources (list of any URI): a list of resource URNs, one for each requested resource.

This function recognizes the hierarchical nature of consents as well as consents granted to Resource Groups. See the rules for granting consent in Accessing Resources by Consent.

Unsupported XACML Features

The Identity Broker supports only those XACML 3.0 features that are necessary for optimized policy decisions. When creating policies, consider that the following XACML 3.0 features are *not* supported:

- **No support for embedded XML content in a request**. In typical XACML deployments, embedded XML content is used to provide additional attributes to an incoming XACML request, needed to make an access decision. For example, these

additional attributes could be captured consent information needed to complete a decision. In terms of security, the policy endpoint would have to add the embedded XML content when formulating the request, rather than just arriving from the requesting application. The Identity Broker solution retrieves these additional attributes directly from the persistent store and does not convert it into XML, to be fully XACML-compliant. As a result, the following XACML elements related to XML processing have *not* been implemented:

- `<PolicyDefaults>` and `<PolicySetDefaults>`
- `<XPathVersion>`
- `<AttributeSelector>`
- `<Content>`
- XPath functions `xpath-node-count`, `xpath-node-equal`, and `xpath-node-match`.

- **No support for Obligations or Advice**. Obligations and Advice are additional actions, specified by policy, which either must (in the case of Obligations) or may (in the case of Advice) be taken by the policy endpoint whenever a particular policy decision is rendered. An example might be to send an email to an individual every time permission is granted to access some resource owned by that person. The following XACML elements related to Obligations and Advice are not implemented:
  - `<Obligations>`
  - `<Advice>`
  - `<AttributeAssignment>`
  - `<ObligationExpressions>` and `<ObligationExpression>`
  - `<AdviceExpressions>` and `<AdviceExpression>`
  - `<AttributeAssignmentExpression>`
- **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 Identity 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 which can be found at `http://docs.oasis-open.org/xacml/3.0/`. The Identity Broker only supports one version of a multiple-

decision request by using multiple `<Attributes>` of the same category in the request. In addition, Identity 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 Identity 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>`

Using Data View Attributes in Policy

Any attribute of the resource owner (specified by `owner-id`) or the current actor (specified by `actor-id`) is available for policy evaluation through access to the Data View associated with the owner or actor. These attributes are accessed by referencing the URN of the Resource object that is generated when the corresponding Data View schema is imported into the

Identity Broker. The XACML attribute category is used to indicate whether the attribute should be retrieved from the owner's record or the actor's record.

For example, the following XACML fragment evaluates to `TRUE` if the resource owner resides in the state of Texas:

```
<Apply FunctionId="urn:oasis:names:tc:xacml:3.0:string-equal-ignore-case">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    TEXAS
  </AttributeValue>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
    <AttributeDesignator
      Category="urn:unboundid:names:1.0:attribute-category:resource-owner"
      AttributeId="urn:scim:schemas:core:1.0:addresses.preferred.region"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
  </Apply>
</Apply>
```

The user's home state is referenced by the SCIM attribute URN `urn:scim:schemas:core:1.0:addresses.preferred.region`. Interest in the owner's address is indicated by the use of the attribute category `urn:unboundid:names:1.0:attribute-category:resource-owner`.

To reference an attribute of the current actor, the attribute category `urn:unboundid:names:1.0:attribute-category:actor` is used.

Policy Sections and Functions Described

The following is the Owned Resource policy, installed with the Identity Broker. 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 Owned Resource Policy

```
[1] <?xml version="1.0" encoding="UTF-8"?>
[2] <Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
[3]         PolicyId="urn:unboundid:policy:Owned Resource Access"
[4]         Version="1"
[5]         RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:den
y-unless-permit">
[6]   <Description>
[7]   Policy to restrict access to 'owned' resources to the resource owner or an actor wi
th a privileged entitlement.
[8]   The policy is only applicable to requests that include both an owner and actor attr
ibute.
[9] </Description>
[10] <Target>
[11]   <AnyOf>
[12]     <AllOf>
[13]       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
[14]         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">.*</Attri
buteValue>
[15]         <AttributeDesignator Category="urn:unboundid:names:1.0:attribute-category:re
source-owner"
[16]                              AttributeId="urn:unboundid:names:1.0:owner:owner-id"
```

```
[17]                                DataType="http://www.w3.org/2001/XMLSchema#string"
[18]                                MustBePresent="false"/>
[19]       </Match>
[20]       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
[21]         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">.*</Attri
buteValue>
[22]         <AttributeDesignator Category="urn:unboundid:names:1.0:attribute-category:ac
tor"
[23]                                AttributeId="urn:unboundid:names:1.0:actor:actor-id"
[24]                                DataType="http://www.w3.org/2001/XMLSchema#string"
[25]                                MustBePresent="false"/>
[26]       </Match>
[27]     </AllOf>
[28]   </AnyOf>
[29] </Target>
[30] <Rule RuleId="urn:unboundid:names:ruleid:isOwner" Effect="Permit">
[31]   <Description>Permit access if the actor is also the owner of the resource.</Descri
ption>
[32]   <Condition>
[33]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[34]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
[35]         <AttributeDesignator Category="urn:unboundid:names:1.0:attribute-category:re
source-owner"
[36]                                AttributeId="urn:unboundid:names:1.0:owner:owner-id"
[37]                                DataType="http://www.w3.org/2001/XMLSchema#string"
[38]                                MustBePresent="true"/>
[39]       </Apply>
[40]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
[41]         <AttributeDesignator Category="urn:unboundid:names:1.0:attribute-category:ac
tor"
[42]                                AttributeId="urn:unboundid:names:1.0:actor:actor-id"
[43]                                DataType="http://www.w3.org/2001/XMLSchema#string"
[44]                                MustBePresent="true"/>
[45]       </Apply>
[46]     </Apply>
[47]   </Condition>
[48] </Rule>
[49] <Rule RuleId="urn:unboundid:names:ruleid:isPrivilegedActor" Effect="Permit">
[50]  <Description>Permit access if the actor holds a privileged entitlement, by default
'broker_admin'.</Description>
[51]   <Condition>
[52]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-mem
ber-of">
[53]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
[54]         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
[55]           broker_admin
[56]         </AttributeValue>
[57]         <!--
[58]          To give this ability to entitlements other than 'broker_admin', add them he
re
[59]         -->
[60]       </Apply>
[61]       <AttributeDesignator Category="urn:unboundid:names:1.0:attribute-category:acto
r"
[62]                            AttributeId="urn:scim:schemas:core:1.0:entitlements"
[63]                            DataType="http://www.w3.org/2001/XMLSchema#string"
[64]                            MustBePresent="false"/>
```

```
[65]     </Apply>
[66]   </Condition>
[67] </Rule>
[68]</Policy>
```

Section Descriptions

Sections are described by line numbers.

- [3] The `PolicyId` specification, must be unique among all policies installed in the Identity Broker.
- [5] The `deny-unless-permit` combining algorithm indicates that if no rule results in a `PERMIT`, then the result of the policy will be `DENY`.
- [6-9] The description here will be displayed when this policy is viewed in the Identity Broker Console.
- [10-29] The `Target` specification for the policy. This policy only is applied to requests that contain both an `actor-Id` and an `owner-Id`, since the purpose of the policy is to protect the owner's data from actors that are not authorized to access it. If the incoming request does not contain both attributes, the policy will return a result of `NOT_APPLICABLE`.
- [13-26] Placing two `Match` elements inside a single `AllOf` element indicates that both Matches must be true for the Target to be matched.
- [13-14] By using the regular expression of `.*`, which matches any string, the `Match` function will be true for any non-empty value of `owner-id`, and false if `owner-id` is not present.
- [30] This rule returns a value of `PERMIT` if the owner and actor are the same person.
- [34] Since `AttributeDesignators` always return a XACML bag, using the `string-one-and-only` function returns the single value from the bag.
- [49] A rule that will return a value of `PERMIT` if the actor possesses a privileged entitlement. This is intended to allow an administrator or customer support representative to have access to any user's data.
- [52] The function `string-at-least-one-member-of acts` on two bags, and return value of `TRUE` if at least one value from the first bag is contained in the second bag. The comparison is case-sensitive.
- [53] The first argument to `string-at-least-one-member-of` is a list of entitlements that should grant privileged access. By default, it contains a single value `broker_admin`.
- [57-59] To customize this policy, additional `AttributeValue` elements can be added here, each containing a privileged entitlement name.

- [61] The second argument to `string-at-least-one-member-of` is the list of entitlements possessed by the actor, obtained from the request context.
- [61-62] The `Category` specification `urn:unboundid:names:1.0:attribute-category:actor` and the `AttributeId` specification `urn:scim:schemas:core:1.0:entitlements` indicate that the policy is interested in the value of the entitlements attribute of the user identified by the actor-id of the current request context.

Managing Policies

The Identity Broker Console enables creating, editing, and managing policies and policy sets. Policies can be generated from a Policy Template in the Identity Broker Console, or by importing a XACML file. Policies that are 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 Identity Broker.
- Pass XACML function checks for the correct number and type of parameters.

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

If the installation option was chosen, the Identity Broker provides several policies that can be customized.

Admin API Policy – Deny Identity Broker administrative access to unauthorized applications or users. By default, access to administrative actions is allowed for a user with the `broker_admin` entitlement and for resources that have the `urn:unboundid:resources:broker_admin` prefix. This is required by the system and can be edited, but should not be deleted. See The Identity Broker Administrative Resources for a list of actions that this policy allows and under what conditions.

Consent Policy – Policy that will return a decision of Permit if the resource owner has consented to allow access to all of the resources in a request. If the resource owner has failed to grant consent for any resource in the request, the policy will return a Deny decision. If the incoming request does not specify a resource owner, the policy decision will be Not Applicable. This is required by the system.

Owned Resource Access Policy – Policy to deny access to resources unless the requester is the owner of the resources or an actor that has a privileged entitlement. This policy is only evaluated for requests that contain both an owner and actor attribute. This is required by the system.

Tag Policy – Policy that will return a decision of Permit if the requesting application holds all governance tags held by all requested resources.

Trust Level Policy – Policy that will return a decision of Permit if the maximum trust level of all resources is less than or equal to the trust level of the requesting application.

User Create and Update Policy – Policy for creating and updating a user through the SCIM endpoint. This is required by the system and should not be deleted.

To Create a New Policy

1. Click **Privacy Policies** and select **Policies**.
2. On the Policies page, click **New Policy**.
3. Enter a name for the new policy.
4. Select the **Enable in Production** box to run this policy, or select it after the policy has been tested.
5. If this policy will be created from a policy template, click in the **Policy Template** field and select an available template.
6. Perform one of the following to select a XACML policy:
    - If creating the policy from a template, the template parameters are loaded in the **Template Parameters** tab. Enter values for each parameter.
    - Click **Browse** to locate and import a XACML file.
    - Create a new policy in the **XACML** field. Each line is numbered. Text can be copied to and from the field. When saved, the policy XACML structure is validated.
7. Click **Save**.

To Edit a Policy

1. Click **Privacy Policies** and select **Policies**.
2. On the Policies page, click the **Edit** button for the item to modify.
3. Change the name for the policy.
4. Change the **Enable in Production** option.
5. Perform one of the following to select a XACML policy:
    - If the policy was created from a template, change values for each parameter.
    - Click **Browse** to locate and import a XACML file.
    - Edit the policy in the **XACML** field. Each line is numbered. Text can be copied to and from the field. When saved, the policy XACML structure is validated.
6. Click **Save**.

To Export a XACML Policy

1. Click **Privacy Policies** and select **Policies**.
2. On the Policies page, click the drop-down arrow next to the **Edit** button in a policy row.
3. Select **Export XACML**.
4. Open or save the file.

To Enable or Disable a Policy in Production

1. Click **Privacy Policies**, and select **Policies**.
2. On the Policies page, click the drop-down arrow next to the **Edit** button in a policy row.
3. Change the production setting.
4. Confirm the action.

To Delete a Policy

1. Click **Privacy Policies** and select **Policies**.
2. On the Policies page, click the drop-down arrow next to the **Edit** button in a policy row.
3. Select **Delete**.
4. Confirm the action.

Managing Policy Sets

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

To Create a Policy Set

1. Click **Privacy Policies** and select **Policy Sets**.
2. On the Policies Sets page, click **New Policy Set**.
3. Enter a name for the new policy set.
4. Select the **Enable in Production** box to run this policy, or select it after the policy has been tested.
5. Click **Browse** to locate and select the policy set (in XACML) to use.
6. Click **Save**.

To Edit a Policy Set

1. Click **Privacy Policies** and select click **Policy Sets**.
2. On the Policies page, click the **Edit** button for the item to modify.
3. On the Edit Privacy Policy dialog, change the name for the policy.
4. Review the Policy Sets for which the policy is a member.
5. Click **Browse** to locate and import a XACML file that represents the set.
6. Click **Save**.

To Export a Policy Set

1. Click **Privacy Policies** and select **Policy Sets**.
2. On the Policy Sets page, click the drop-down arrow next to the **Edit** button in a row.
3. Select **Export XACML**.
4. Open or save the file to a new location.

To Disable a Policy Set

1. Click **Privacy Policies** and select **Policy Sets**.
2. On the Policy Sets page, click the drop-down arrow next to the **Edit** button in a row.
3. Change the production option.
4. Confirm the action.

To Delete a Policy Set

1. Click **Privacy Policies** and select **Policy Sets**.
2. On the Policies Sets page, click the drop-down arrow next to the **Edit** button in a row.
3. Select **Delete** and confirm the action.

Managing Policy Sandboxes

A policy sandbox is an isolated policy engine (non-production) that is useful for testing new or modified policies or policy sets.

The Identity Broker provides an OAuth Consent Evaluation sandbox that is used by the OAuth 2.0 authorization endpoint to determine whether to prompt a user for consent of requested scopes, in isolation from other policies. To customize consent policy, modify or replace the policy in this sandbox.

To Create a New Policy Sandbox

1. Click **Privacy Policies** and select **Policy Sandboxes**.
2. On the Policies Sandboxes page, click **New Policy Sandbox**.
3. On the **General** tab, enter a name for the new policy sandbox and optional description.
4. In the **Policies and Policy Sets** drop-down list, select one of the following options:
    - **Include Only Specified Policies and Policy Sets**. The sandbox will provide a decision based only on the policies that are added to this sandbox.
    - **Include All Policies and Policy Sets Enabled in Production**. When selected, this option enables adding new policies to the sandbox or removing existing

policies from the sandbox. New policies are tested in combination with the existing policies.

5. In the Policy Combining Algorithm list, select the combining algorithm for this sandbox.
6. Click either the **Policies** tab or the **Policy Sets** tab.
7. Search for and select policies to add to the sandbox.
8. Click **Save**.

To Edit a Policy Sandbox

1. Click **Privacy Policies** and select **Policy Sandboxes**.
2. On the Policies Sandboxes page, click the **Edit** button for the item to modify.
3. On the **General** tab, change the name and optional description.
4. In the **Policies and Policy Sets** drop-down list, select one of the following options:
   - **Include Only Specified Policies and Policy Sets**.
   - **Include All Policies and Policy Sets Enabled in Production**. If this option is selected, additional tabs display that enable including or excluding policies from all that are enabled in production.
5. In the Policy Combining Algorithm list, select the combining algorithm for this sandbox.
6. Click the **Policies** tab.
7. Search for and select policies to add to the sandbox.
8. Click the **Policy Sets** tab.
9. Search for and select policy sets to add to the sandbox.
10. Click **Save**.

To Run a Policy Test

1. Click **Privacy Policies** and select **Policy Sandboxes**.
2. On the Policy Sandboxes page, click the drop-down arrow next to the **Edit** button in a row.
3. Select **Run Test**. (Policy tests must already be created.)
4. Select the test to run in this sandbox and click **Run Test**. The test results are shown in the Decision box. To view a trace of the Policies or Policy Sets test, click **Show Details**.

To Delete a Policy Sandbox

1. Click **Privacy Policies** and select **Policy Sandboxes**.
2. On the Policy Sandboxes page, click the drop-down arrow next to the **Edit** button in a row.
3. Select **Delete**.
4. Confirm the action.

Managing Policy Templates

A policy template is a parameterized XACML policy that is used to create new policies. A single template can be used to create multiple policies. A template becomes a policy instance when it is used to create a policy and attribute values are specified. Attribute values are based on data types defined in *OASIS Committee Specification 01, eXtensible access control markup language 193 (XACML) Version 3.0,* and a string that represents the value of the data type, such as:

```
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string>
    string_value
</AttributeValue>
```

Policies generated from a template are linked to that template to easily determine how and from what source a policy was generated. The link between the template and the policies is simply for reference. Once a template is used to create policies, importing a new XACML definition or changing any of the parameter IDs will break the link between the template and the policies.

To Import a New Policy Template

A sample template (`IPAddressTemplate.xml`) is available in the resources directory of the Identity Broker distribution. It can be modified and imported.

1. Click **Privacy Policies** and select **Policy Templates**.
2. Click **New Policy Template**. The Create Policy Template dialog opens.
3. In the **Name** box, enter a name for the new policy template.
4. In the XACML file box, browse for the template to import.
5. Click **Save** to store it.

To Edit a Policy Template

1. Click **Privacy Policies** and select **Policy Templates**.
2. On the Policy Templates page, click the **Edit** button for the item to modify.

3. On the **General** tab, change the name for the policy.
4. To import a XACML definition, click **Browse** to locate and import the XACML file.

   Importing a new XACML file breaks the link between this template and any policies that were generated from it. Template parameters cannot be edited for policies that do not have a template link.
5. On the **Parameters** tab, review the parameters for the policy.
6. Click **Save**.

To Export a XACML Policy Template

1. Click **Privacy Policies** and select **Policy Templates**.
2. On the Policy Templates page, click the drop-down arrow next to the **Edit** button in a row.
3. Select **Export XACML**.
4. Open or save the file to a new location.

To Delete a Policy Template

Deleting a template does not impact the policies that were created from it, but the policies will no longer have the template reference.

1. Click **Privacy Policies** and select **Policy Templates**.
2. On the Policies Templates page, click the drop-down arrow next to the **Edit** button in a row.
3. Select **Delete**.
4. Confirm the action.

Managing Policy Tests

Policy Tests run policy scenarios to ensure that they work as designed before deploying in production. The Policy Tests create an authorization request and specifies the client application 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.

To Create a New Policy Test

1. Click **Privacy Policies** and select **Policy Tests**.
2. Click **New Policy Test**. The Create New Policy Test dialog opens.
3. In the **Name** box, enter a name for the new policy test.

4. Enter an optional description in the **Description** box.
5. Click the **Requester** tab.
6. Enter or select the application making the request in the **Subject** box. This should be an application registered with the Identity Broker server.
7. Select the type of action that the requestor will be conducting from the **Action** list. Options are Create, Delete, Read, or Update.
8. Select the purpose of the data access from the **Purpose** list.
9. Enter an optional resource owner. This is the unique SCIM ID as stored in the backend user store.
10. Enter an optional actor. This must also be a unique SCIM ID as stored in the backend user store.
11. Click the **Resources** tab or **Resources Group** tab.
12. Select the resources or the group of resources that will be accessed by the requestor application.
13. Click the **Additional Arguments** tab and add any optional arguments for this test.
14. Click the **Run Test** tab, and select the Policy Sandbox to run the test against.
15. Click **Run Test**. The Identity Broker Console arrives at a decision and lists the total processing time. To view a trace of the decisions, click **Show Details**.
16. Click **Save**.

To Edit a Policy Test

1. Click **Privacy Policies** and select **Policy Tests**.
2. Click **Edit** for the policy test to edit.
3. In the **Name** box, change the policy test name.
4. Enter or change an optional description in the **Description** box.
5. Click the **Requester** tab.
6. Change the subject (application) making the request in the **Subject** box.
7. Change the type of action that the requestor will be conducting from the **Action** list. Options are Create, Delete, Read, or Update.
8. Select or change the purpose of the data access from the **Purpose** list.
9. Enter an optional resource owner. This is the unique SCIM ID as stored in the backend user store.
10. Enter an optional actor. This must also be a unique SCIM ID as stored in the backend user store.

11. Click the **Resources** tab or **Resources Group** tab.
12. Select the resources or the group of resources that will be accessed by the requestor application.
13. Click the **Additional Arguments** tab and add or change any optional arguments for this test.
14. Click the **Run Test** tab and select or change the Policy Sandbox to run the test against.
15. Click **Run Test**. The Identity Broker Console arrives at a decision and lists the total processing time. To view a trace of the policy decisions, click **Show Details**.
16. Click **Save**.

To Delete a Policy Test

1. Click **Privacy Policies** and select **Policy Test**.
2. On the Policy Test page, click the drop-down arrow next to the **Edit** button in a row.
3. Select **Delete**.
4. Confirm the action.

Chapter 6: Monitoring the Identity Broker

The Identity Broker's server performance, consent traffic, client application access, and policy activities can be monitored and displayed on the Metrics page. The Metrics section of the Identity Broker Console displays data from a configured Metrics Engine instance, providing a real-time picture of identity data.

See the *UnboundID Identity Broker Installation Guide* and the *UnboundID Identity Broker Administration Guide* for details about configuring the Identity Broker to display Metrics Engine data.

This section explains how to monitor the Identity Broker. In this section, the following tasks are performed:

Dashboards and Metrics

Dashboards are configured during the Metrics Engine installation and display data gathered by the Metrics Engine. Data includes:

- Authorizations granted and denied to client applications.
- Consents granted, denied, and abandoned by customers.
- Most requested data.
- Most requesting client applications.

See the *UnboundID Metrics Engine Administration Guide* and the *UnboundID Identity Broker Installation Guide* for steps to configure dashboards.

About System and Consent Metrics

The Metrics page contains several charts to measure and monitor Identity Broker system and user consent activity. The following categories can be made available on the Metrics page:

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.

The data that displays in the Identity Broker Console is dependent on the data configured for the Metrics Engine. Dashboards are configured from the Metrics Engine server.

To Change Metrics Data

No data is displayed until it is generated by the Metrics Engine. Both the Metrics Engine and the Identity Broker server must be configured to display data. See the *UnboundID Metrics Engine Administration Guide* and the *UnboundID Identity Broker Installation Guide* for steps to configure dashboards.

1. Click **Metrics**. The dashboards configured for the system are displayed. Mouse over data in any chart to see additional details.
2. To change the data displayed, click the **Metrics** tool button.
3. In the **Instance** drop-down menu, choose the Identity Broker instance from which data is displayed.

4. In the **Number of Points** drop-down menu, choose the number of data points to display on each chart.
5. Select either the **Time Range** or **Start Time** option.
    - For **Time Range** option, select a range from the drop-down menu and the **Time Offset**. Time offset is the time at which the range should start. For example, if an 8-week range and a 5-day offset are both selected, the data will display for 8 weeks starting 5 days prior to the current date.
    - For the **Start Time** option, select a start and end time from the drop-down menus.
6. Click **Apply**. Data displays based on the options selected.

Chapter 7: Testing

After all Identity Broker components are configured, test them to make sure that policies and consent actions work as expected.

This section explains how to test Identity Broker components prior to production. In this section, the following tasks are performed:

Testing Sample Policies

Testing the OAuth 2.0 Authorization Flows

Troubleshooting Policies with Traces

Configuring the Policy Debug Logger

Configuring the File-Based Authorization Logger

Testing the Sample Policies

If sample data was installed during the setup process, four sample Policy Tests are available:

- External App Request for Billing History
- External App Request for Customer Profile
- Internal App Request for Billing History
- Internal App Request for Consumer Data

To Test the Sample Policies

1. Log into the Identity Broker Console.
2. Click **Privacy Policies**, and select **Policy Tests**.
3. On the Policy Tests page, click the arrow next to the **Edit** button for the External App request for Billing History, and then select **Run Test**. In this example, the `ExternalAppTwo` is requesting to see the Billing History resource for the `sampleuser1`, who has given consent to let the application view his billing history.
4. On the Policy Test dialog, click **Run Test**.
5. Run the other Policy Tests to view the decision.

Testing the OAuth2 Authorization Flows

The Identity Broker provides a command-line tool to test OAuth 2.0 authorization flows.

To Test the OAuth2 Client Credentials Grant Type

The `oauth2-request` tool can also be used to retrieve an access token using the OAuth 2.0 client credentials grant type. This grant type is used when the requesting application would like to access its own resources, so a separate set of owner credentials is not needed.

Note

Refresh tokens cannot be requested using the client credentials grant type.

1. Obtain the token from client credentials.

```
$ broker1/bin/oauth2-request token-from-client-credentials \
--clientID a07bfe59-124a-4747-9037-e3e099b68203 \
--clientSecret 1RMGfNz38e --displayToken


Parameter     Value(s)
----------------------------------------------------------------------------
```

```
Access Token  MC2AAQGBBmg2OEVvUYIg-XlYUSaBC4BxLoeOEwbl-CcC66tRPYvtWHo9-TxhyxM
Token Type    bearer
Scope         urn:com.example.scope.test_resource
Expires In    11 hours, 59 minutes, 58 seconds
Expiration    20130419093824Z
```

2. Validate the token.

```
$ broker1/bin/oauth2-request validate-token \
--clientID a07bfe59-124a-4747-9037-e3e099b68203 \
--clientSecret 1RMGfNz38e \
--token MC2AAQGBBmg2OEVvUYIg-XlYUSaBC4BxLoeOEwbl-CcC66tRPYvtWHo9-TxhyxM

Parameter   Value(s)
-----------------------------------------------
Client ID   a07bfe59-124a-4747-9037-e3e099b68203
Scope       urn:com.example.scope.test_resource
Issued At   20130418213824Z
Expires In  11 hours, 59 minutes, 8 seconds
```

To Test the OAuth2 Auth Code and Implicit Grant Types

The authorization code and implicit grant types are intended to be used interactively with a web browser. The `oauth2-request` command line tool does not support those grant types. The the Sample Sign-In application, included with the Identity Broker, provides a way to test these grant types. See the *UnboundID Identity Broker Application Developer Guide* for information about this application.

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 results can be difficult. To aid in the debugging process, the Identity Broker supports the ability to generate a trace of any policy decision. Tracing is automatically enabled when using the Policy Test feature, with the trace output displayed in the Identity Broker Console. It can also be enabled for all policy decisions by Configuring the Policy Debug Authorization Logger.

A policy 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.

When using the Policy Debug Authorization Logger, by default policy traces are written to the log file `<server-root>/logs/policy-decision-trace`. The following is an excerpt from a trace entry:

```
[22/Sep/2014:12:59:09.968 -0500] POLICY TRACE threadID=14244
application="UnboundID Profile Manager Sample"
owner="9f8a23-659acbff-f234-305e-8776-2dc725d4899f"
actor="9f8a23-659acbff-f234-305e-8776-2dc725d4899f"
action="Read" purpose="ANY"
resources="urn:scim:schemas:core:1.0:emails.preferred.value"
trace=<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DecisionTrace xmlns:ns2="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
<PolicySet id="urn:unboundid:policyset:PDP-global" combiningAlgId="deny_overrides" result
="DENY">
```

The following are included:

- The first line of the log entry identifies the message type as `POLICY TRACE`.
- The parameters of the XACML request being traced are listed, including the application, resource owner, actor, action, purpose, and resources.
- Following this is the trace itself, which is included in the `<DecisionTrace>` XACML element.
- Following the `<DecisionTrace>` element is a list of policy sets and policies that were consulted in order to come to a decision. The first policy set is the `PDP-global` set, which is an internally-generated set that consists of all top-level policies and policy sets that are enabled in the Identity Broker.
- The `combiningAlgId` attribute corresponds to the global combining algorithm that was chosen for the Policy Service. (See the *UnboundID Identity Broker Installation Guide* for information about configuring the Policy Service.)
- The `result` XML attribute specifies the final result of the policy engine's evaluation of the request, which in this example is `DENY`.

The trace then includes entries for each policy, rule, and target evaluated during the decision process. For example, the next element might include the following:

```
<Policy id="urn:unboundid:policy:UserCreateAndUpdate"
combiningAlgId="deny_unless_permit" result="NOT_APPLICABLE">
  <Target result="NoMatch">
    <Match function="string_equal" result="NoMatch">
      <Argument type="AttributeValue" value="Create"/>
      <Argument type="AttributeDesignator" value="Read"/>
    </Match>
    <Match function="string_equal" result="NoMatch">
      <Argument type="AttributeValue" value="Update"/>
      <Argument type="AttributeDesignator" value="Read"/>
    </Match>
  </Target>
</Policy>
```

This trace element shows that the first policy evaluated was the User Create And Update policy, and that it returned a result of `NOT_APPLICABLE`. The sub-elements of this policy element show that the result of the policy's `Target` specification was `NoMatch`.

The NoMatch result is due to both child Match elements also resulting in NoMatch. The request attribute being compared (action-id) was not equal to either "Update" or "Create."

Each AttributeValue and AttributeDesignator that was evaluated is shown with the value that was received by the policy engine when the policy was evaluated. Since the trace does not include the name of the attribute that is referenced by an AttributeDesignator, it may be helpful to reference a copy of the policy itself while analyzing the trace output.

The next step in the analysis might be to look for the trace element of the policy that caused the request to be denied. Consider the following snippet which could be from the same trace entry:

```
<Policy id="urn:unboundid:policy:GovernanceTagPolicy" combiningAlgId="first_applicable" r
esult="DENY">
  <Target result="Match"/>
  <Rule id="urn:unboundid:names:1.0:ruleid:missing-request-governance-tags" effect="DENY"
result="DENY">
    <Condition result="TRUE">
      <Apply function="and" result="true">
        <Argument type="Apply" value="true">
          <Apply function="integer_equal" result="true">
            <Argument type="VariableReference" value="0" variableId="requestTagsSize">
              <Apply function="string_bag_size" result="0">
                <Argument type="AttributeDesignator" value="[]" variableId="urn:unboundid
:names:1.0:subject:governance-tags"/>
              </Apply>
            </Argument>
          <Argument type="AttributeValue" value="0"/>
        </Apply>
      </Argument>
      <Argument type="Apply" value="true">
        <Apply function="integer_greater_than" result="true">
          <Argument type="VariableReference" value="1" variableId="resourceTagsSize">
            <Apply function="string_bag_size" result="1">
              <Argument type="AttributeDesignator" value="[EmailTag]" variableId="urn:unb
oundid:names:1.0:resource:governance-tags"/>
             </Apply>
            </Argument>
            <Argument type="AttributeValue" value="0"/>
          </Apply>
         </Argument>
       </Apply>
     </Condition>
   </Rule>
</Policy>
```

The first line shows that GovernanceTagPolicy returned a result of DENY. It also shows that only a single rule of the policy was evaluated, urn:unboundid:names:1.0:ruleid:missing-request-governance-tags,and that this rule's result was DENY.

This result caused policy evaluation to be terminated since the policy's combining algorithm is first_applicable, which is defined by XACML to terminate once a rule returns a result other than NOT_APPLICABLE. This particular rule checks if the requested Resource is associated with Tags that are not associated with the requesting application. It does so by checking whether

the Resource has a greater number of Tags than the application does. This is done by applying the XACML function `integer_greater_than` to two arguments which are:

- the `string_bag_size` of resource Tags.
- the `string_bag_size` of subject Tags.

In the trace above, the count of Resource Tags is `1` while the count of Subject tags is `0`. It also shows that the Tag associated with the resource is named `EmailTag`. This is the reason that the authorization request was denied.

Configuring the Policy Debug Authorization Logger

The Policy Debug Authorization Logger is used to analyze or troubleshoot policy decisions in a production environment. Policy Tests automatically trace all policy decisions and the trace is returned as part of the result. The logger is disabled by default. Enabling the logger causes all policy decisions to be traced and stored in the `/logs` directory.

The logger should be used for a limited amount of time and then disabled, especially in a production environment. A large amount of data can be generated and stored if the Identity Broker is busy.

A trace filter can be used to restrict the decision traces that are logged. A trace filter is a standalone XACML "Target" specification and can be created to filter the logging to only requests that apply to the trace filters. A trace filter can be created with the `broker-admin create-trace-filter` tool. See Identity Broker Tools for more information about the `broker-admin` command.

To configure the logger:

1. Run the following command on the Identity Broker server:

```
$ bin/dsconfig set-log-publisher-prop --publisher-name "Policy Debug
  Authorization Logger" --set enabled:true
```

2. Enter the Identity Broker hostname or IP address and press **Enter**.
3. Specify the option to connect to the Identity Broker and press **Enter**.
4. Enter the connection port, or press **Enter** to confirm the default.
5. Enter the administrator user bind DN, or press **Enter** to confirm the default (`cn=Directory Manager`).
6. Enter the password for this account and press **Enter**.
7. Choose to update all servers or only the current server. The Configuration Menu is displayed:

```
>>>> Configure the properties of the File Based Authorization Log Publisher
Property                  Value(s)
--------------------------------------------------------------------------------
```

```
1)   description           "When this logger is enabled, detailed debugging
                            information about policy decisions will be written
                            to the specified log file. The output can be verbose,
                            so be careful using this on a server in production
                            or under high load."

2)   enabled                true
3)   logged-message-type    decision-trace
4)   timestamp-precision    milliseconds
5)   log-file               logs/policy-decision-trace
6)   log-file-permissions   640
7)   append                 true
8)   rotation-policy        Size Limit Rotation Policy
9)   retention-policy       File Count Retention Policy, Free Disk Space
                            Retention Policy
10)  sign-log               false
```

8. Make any necessary changes.
9. Disable the logger. It should not stay enabled for a long period.
10. Review the policy decisions.

Configuring the Authorization Logger

The File-Based Authorization Logger records all policy, authorization, and consent decisions made by the Identity Broker, including:

- OAuth 2.0 events, such as access token grants and revocations.
- User consent grants and revocations.
- Policy decisions and request parameters.
- External identity provider events, such as tokens granted, linking events, and attributes retrieved.

The File-Based Authorization Logger is enabled by default, and is configured with the `dsconfig` tool. Logs are stored in `logs/authorization` by default. For additional information about UnboundID log publishers, see the *UnboundID Identity Data Store Administration Guide*.

Note Custom policy request parameters can be added to this or the Policy Debug Authorization Logger through the Custom Logged Authorization Request Attribute parameter, which is an advanced `dsconfig` option.

Chapter 8: System Administration

The Identity Broker provides command-line tools to perform system administrative tasks.

This section explains the Identity Broker Console commands and tools, and other administration options. In this section, the following tasks are performed:

Identity Broker Configuration Tools

The command-line tools are located in the `<server-root>/bin` directory. The `broker-admin` tool provides most of the same functionality as the Identity Broker Console. Each command-line tool provides help options with examples. List all commands using the `--help` argument, all sub-commands using the `--help-subcommands` argument, and a detailed help for a single subcommand using the `--help` argument with the subcommand name.

```
$ bin/broker-admin --help
$ bin/broker-admin --help-subcommands
$ bin/broker-admin update-policy-template --help
```

The following tools manage the various Identity Broker administrative tasks. A full list of tools is available in Identity Broker Tools.

- **broker-admin** – Runs administrative operations. Use this tool to create and configure applications, policies, resources, tags, and trust levels. All of these actions can also be done in the Identity Broker Console.
- **consent-admin** – Runs consent management operations. Use this tool to add consents, list consent history, list applications and resources for which consent was granted, and revoke consent.
- **evaluate-policy** –Requests a policy decision from the Identity Broker. Use this tool to view policy decisions including a decision trace in XACML format.
- **oauth2-request** – Tests token functions of the Identity Broker. Use this tool to manage OAuth2 tokens on behalf of a registered application.
- **dsconfig** – Provides additional configuration options for the Identity Broker environment. This tool provides an interactive, menu-driven mode to facilitate tasks such as adding additional user stores.
- **prepare-external-store** – Prepares the external data stores for the Identity Broker. This is run as part of the `create-initial-broker-config` tool during installation, but can be used to update the Broker Store or an external user store.
- **collect-support-data** – Collects system information useful in troubleshooting problems. The information is packaged as a zip archive.

All Identity Broker Tools

Available Identity Broker tools are:

Identity Broker Tools

Tool	Description
backup	Run full or incremental backups on one or more Identity Brokers. This utility also supports the use of a properties file to pass predefined command-line arguments.

Tool	Description
base64	Encode raw data using the base64 algorithm or decode base64-encoded data back to its raw representation.
broker-admin	Invoke administrative operations over the Identity Broker REST API.
collect-support-data	Collect and package system information useful in troubleshooting problems. The information is packaged as a ZIP archive that can be sent to a technical support representative.
consent-admin	Manage a resource owner consent over the Identity Broker REST API. Consent is authorized by a resource owner to allow access to resources by an application.
config-diff	Generate a summary of the configuration changes in a local or remote server instance. The tool can be used to compare configuration settings when troubleshooting issues, or when verifying configuration settings on new servers.
create-initial-broker-config	Create an initial Identity Broker configuration.
create-rc-script	Create a Run Control (RC) script that can be used to start, stop, and restart the Identity Broker on Unix-based systems.
dsconfig	View and edit the Identity 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 Identity 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 Identity Broker.
ldapmodify	Perform LDAP modify, add, delete, and modify DN operations in the Identity Broker.
ldappasswordmodify	Perform LDAP password modify operations in the Identity Broker.
ldapsearch	Perform LDAP search operations in the Identity 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 Identity 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 Identity Broker. Any added extensions require a server re-start.
oauth2-request	Performs OAuth 2.0 requests on the Identity Broker. This tool can be used to test OAuth 2.0 functions of the Identity Broker, and to manage OAuth 2.0 tokens on behalf of registered applications.
prepare-external-store	Prepares the external data stores for the Identity Broker. This is run as part of the `create-initial-broker-config` tool during installation. This tool creates the broker user account, sets the correct password, and configures the account with required privileges. It will also install the necessary schema required by the Identity Broker. This tool can also be used to update (with the `--update` option) an external Broker Store or a data store schema.
remove-defunct-server	Removes a permanently unavailable Identity Broker after it has been removed from

Tool	Description
	its topology by the `uninstall` tool.
restore	Restore a backup of the Identity Broker.
review-license	Review and/or accept the product license.
sample-data-loader	Install or remove sample data for Identity Broker testing and demonstration.
server-state	View information about the current state of the Identity Broker processes.
start-broker	Start the Identity Broker.
status	Display basic server information.
stop-broker	Stop or restart the Identity Broker.
sum-file-sizes	Calculate the sum of the sizes for a set of files.

About the Tools Authentication Arguments

The Identity Broker's tools require various authentication IDs that are different from arguments used with the Identity Data Store. The following table illustrates the differences:

Authentication Arguments for the Identity Broker's Tools

Arguments	Purpose	Used by
`--authClientID` `--authClientSecret`	The client ID and secret of the internal `@BrokerCLI@` application, needed by the command-line tools to obtain a bearer token to read Broker Store configuration information from the Identity Broker REST API. Typically, these arguments do not need to be provided, as the tool will read the required values from the `oauth-admin-client-id` and `oauth-admin-client-secret` properties of the OAuth Service configuration.	`broker-admin`, `consent-admin`, and `evaluate-policy`
`--authID` `--authPassword`	Credentials for the user or administrator running the tool, also needed by the tool itself to obtain a bearer token to read Broker Store configuration information from the Identity Broker REST API. This user must exist in the Identity Broker's user store or Broker Store.	`broker-admin`, `consent-admin`, and `evaluate-policy`

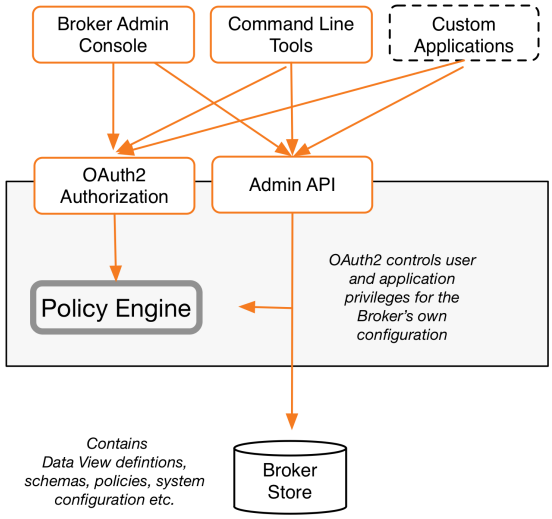
`--application`
`--clientID`
`--clientSecret`
	The credentials of the client application requesting access to a resource.	`oauth2-request`

`--ownerID` `--ownerPassword`	The resource owner's credentials.	`oauth2-request`

Administrative Access

Access to the Identity Broker Console, command-line tools, and Broker Admin API are governed by policy. The default administrative policy, Admin API, defines the account entitlement required for access, the applications that can be used, and the resources that are

available to a user with the required entitlement. This policy can be used as a template to create a set of more granular policies.



Admin API Flow

A call to the Broker Admin API requires the correct scopes or resources needed to perform an administrative task. All of the required resources and scopes needed to manage the Identity Broker are listed in the Identity Broker Console. Administrative resources and administrative scopes cannot be edited or deleted. However, new scopes can be created for a set of actions and then defined in a policy for an administrator or set of administrators to access.

Adding Additional Administrative Accounts

Administrative access is governed by policy and requires the `broker_admin` entitlement, which by default enables access to all Identity Broker tasks. The default Admin API policy allows access to any account with the `broker_admin` entitlement and for any requested resource with the `urn:unboundid:resources:broker_admin` prefix. See Identity Broker Administrative Resources for a full listing.

If this environment needs multiple administrators with finer-grained access to Identity Broker tasks, administrative accounts can be added with custom entitlements such as `manage_applications`. The Admin API Policy can then be updated (or a new policy created) to accept a user ID with that entitlement for a specific set of Identity Broker administrative scopes or resources. See Sample for Adding an Administrator.

By default, an initial administrative account is created by the `create-initial-broker-config` tool and stored in the Broker Store as an LDAP entry in the `ou=Admins,ou=Identity Broker,<base DN>` branch. A user's entitlements are stored in the native data store attribute that is mapped to the Data View attribute `urn:scim:schemas:core:1.0:entitlements.value`. An administrative account can be exposed through any Data View.

Adding an Account to an LDAP Data Store

From the Identity Data Store that acts as the Broker Store, use `ldapmodify` to add a user entry, including a value for the `ds-broker-admin-privilege` attribute:

```
$ bin/ldapmodify

dn: uid=admin-2,ou=Admins,ou=Identity Broker,dc=example,dc=com
changeType: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetorgperson
description: This is the administrative user #2 for the UnboundID Identity Broker
uid: admin
cn: Broker
sn: Admin
userPassword: password
ds-broker-admin-privilege: broker_admin

# Processing ADD request for uid=admin-2,ou=Admins,ou=Identity Broker,dc=example,dc=com
# ADD operation successful for DN uid=admin-2,ou=Admins,ou=Identity Broker,dc=example,dc=
com
```

Adding an Account through a SCIM Front End

In this example HTTP request/response pair, the client is using a bearer token that was granted to an application that is explicitly allowed to create users through a Data View by the User Create and Update policy. The LDAP store adapters that back this Data View have been explicitly configured to allow user creation requests. This is not allowed by default.

```
POST /scim/Users HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate, compress
Authorization: Bearer MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-vzKYVltlykXrZE2AG0F3J3mQjU
YOSP3dCOaIeYEUWSK-OgWfYpuFNpqfGQv91vcvarYUJa44n1srvcYC7yDIoo
Content-Length: 383
Content-Type: application/json; charset=utf-8
Host: example.com:443
User-Agent: HTTPie/0.7.2

{
  "emails": [
     {
        "type": "preferred",
        "value": "bill.lumbergh@example.com"
     }
   ],
  "entitlements": [
     {
        "value": "broker_admin"
     }
   ],
  "name": {
```

```
       "familyName": "Lumbergh",
       "formatted": "Bill Lumbergh",
       "givenName": "Bill"
  },
  "schemas": [
       "urn:scim:schemas:core:1.0"
  ],
  "userName": "bill.lumbergh"
}

HTTP/1.1 201 Created
Content-Length: 507
Content-Type: application/json
Location: https://example.com/scim/v1/Users/9f8a23-a762e2cf-0c61-4994-9a62-bd22a7c4cc77
Server: Jetty(8.1.12.v20130726)

{
  "emails": [
     {
        "type": "preferred",
        "value": "bill.lumbergh@example.com"
     }
   ],
  "id": "9f8a23-a762e2cf-0c61-4994-9a62-bd22a7c4cc77",
  "meta": {
     "created": "2013-11-14T00:35:12.658Z",
     "lastModified": "2013-11-14T00:35:12.658Z",
     "location": "https://x2250-01:1443/scim/v1/Users/9f8a23-a762e2cf-0c61-4994-9a62-bd22
a7c4cc77"
  },
  "name": {
     "familyName": "Lumbergh",
     "formatted": "Bill Lumbergh",
     "givenName": "Bill"
  },
  "schemas": [
     "urn:unboundid:oidc:1.0",
     "urn:scim:schemas:core:1.0"
  ],
  "userName": "bill.lumbergh"
}
```

Sample for Adding an Administrator

The Identity Broker is installed with a single administrative account that has unlimited access to Identity Broker functions. Additional administrators can be added to manage sub-sets of Identity Broker tasks, such as managing policies or applications.

The following example outlines the process for adding an administrator, with a special entitlement, to the Admin API policy with access to just the Data Requestor tasks (applications, application groups, actions, and purposes). The policy is configured to `deny-unless-permit`. Adding this new rule with a specified entitlement will permit an administrator with that entitlement access to the specified resources, while denying access to the other administrative resources.

1. Create an account with a new entitlement called `manage_data_requestors`. See Adding Additional Administrative Accounts.
2. From the Identity Broker Console, export the Admin API policy `AdminAccess.xml` and save it.
3. Create a new rule for the policy and specify that a user with `manage_data_requestors` entitlement can have access to the administrative resources for applications, actions, and purposes in the Identity Broker Console and command-line interfaces:

   `urn:unboundid:resources:broker_admin:applications`

   `urn:unboundid:resources:broker_admin:actions`

   `urn:unboundid:resources:broker_admin:purposes`

   The rule will contain the following, with the resources, applications, and entitlement specifications in bold:

```
<Rule RuleId="urn:unboundid:names:1.0:dataRequestorAccess" Effect="Permit">
  <Description>
   Allow administration of Applications, Actions, and Purposes to
   applications explicitly listed by this rule and users with the
   manage_data_requestors entitlement.
  </Description>
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal"/>
        <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:
                                attribute-category:resource"
                             AttributeId="urn:oasis:names:tc:xacml:1.0:
                                resource:resource-id"
                             DataType="http://www.w3.org/2001/XMLSchema#anyURI"
                             MustBePresent="false" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-bag">
          <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#anyURI">
 urn:unboundid:resources:broker_admin:applications
          </AttributeValue>
          <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#anyURI">
 urn:unboundid:resources:broker_admin:actions
          </AttributeValue>
          <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#anyURI">
 urn:unboundid:resources:broker_admin:purposes
          </AttributeValue>
        </Apply>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
only">
          <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:
                                   subject-category:access-subject"
                               AttributeId="urn:oasis:names:tc:xacml:1.0:
                                   subject:subject-id"
                               DataType="http://www.w3.org/2001/XMLSchema#string"
```

```
                              MustBePresent="false"/>
       </Apply>
       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
UnboundID Broker Admin Console
         </AttributeValue>
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
UnboundID Broker CLI Tools
         </AttributeValue>
        </Apply>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
         string-at-least-one-member-of">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
manage_data_resources
          </AttributeValue>
        </Apply>
        <AttributeDesignator Category="urn:unboundid:names:1.0:attribute-
category:actor"
                             AttributeId="urn:scim:schemas:core:1.0:entitlements"
                             DataType="http://www.w3.org/2001/XMLSchema#string"
                             MustBePresent="false"/>
       </Apply>
    </Apply>
  </Condition>
</Rule>
```

4. Import the policy in the Identity Broker Console.
5. Create a scope that includes just the Data Requestor resources in the Identity Broker Console. The scope cannot contain any additional resources that are not present in the rule, or all resources will be denied.
6. Add that scope to the Identity Broker Console application and the Identity Broker CLI Tools with the `broker-admin` command:

```
$ broker-admin set-application-prop \
  --name "UnboundID Broker Admin Console" \
  --add scopeIds:name=<Data-Requester-Scope>
```

```
$ broker-admin set-application-prop \
  --name "UnboundID Broker CLI Tools" \
  --add scopeIds:name=<Data-Requester-Scope>
```

Application Access to the Identity Broker Admin API

For a client application to be granted access to the Identity Broker Admin API and resources managed by it, the following must be true:

- The application must use a bearer token with one of the administrative scopes defined for the Identity Broker. See Administrative Scopes and Resources.
- The application must be listed in the Admin API policy.

Access to the Admin API is determined by the application that is requesting access, and who is currently logged into that application. By default, a user must have the `broker_admin` entitlement to access administrative resources, and the application through which the user is requesting access must be listed in the Admin API policy. If the application is listed, access is further limited by the authorities/privileges that are possessed by the user logged into the application.

Third-Party Applications

Third-party applications can be used to manage data in the Broker Store. To develop an application that can access the Broker Admin API, the application must be registered with the Identity Broker through the Console or the `broker-admin` tool.

The application must also be present in the list of approved API applications in the Admin Access policy (stored in `resource/AdminAccess.xml`). Add the application by name by appending a new `AttributeValue` element to the `applicationsAllowedAdminAccess` rule:

```
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
   My REST API application
</AttributeValue>
```

Like the Identity Broker applications, third-party applications must request access to an administrative scope when authorizing users to access to the REST API. See Identity Broker Scopes for a list of administrative scopes.

Managing the Broker Web Applications

By default, Identity Broker applications are deployed through an embedded Jetty servlet container. The applications are designed with the Velocity Template Language and can be customized.

Note

Because the Identity Broker Console and sample applications both use the Identity Broker server for authentication, logging into both applications from the same browser (using different tabs) can cause authorization errors. If logging into multiple applications at the same time, use different browsers to keep the sessions and cookies separate. If bookmarking the application pages, bookmark the Identity Broker Console and sample application landing pages, not the login page. Bookmarking the login page causes the same authentication errors.

The Profile Manager Application

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

- Viewing consent for third-party access to the end-user's resources (typically from a web site).
- Revoking consent from client applications.

The Sign-In Sample Application

A sample client application is installed with the Identity Broker Server. It can be used as a model for a client application using the OpenID Connect `/userinfo` endpoint. The application provides the OAuth 2.0 implicit grant flow of an end user signing into the Identity Broker, the Identity Broker prompting the end user for consent to access resources, and the application retrieving the information that is configured in the UserInfo Claims Map on the Identity Broker Server.

Configuring the Broker Login and Consent Pages

The Identity Broker exposes Velocity pages through an HTTP Servlet Extension. Templates are located in:

`/<Broker_home>/config/velocity/templates`

Account registration and password recovery require server configuration. See User Account Registration and Recovery.

login.vm – Defines the Identity Broker log in page and includes icons for external identity provider login. A registration form is also provided for users to create an account through the external identity provider login.

approve.vm – Defines the OAuth approval page presented to end users who need to approve access to resources. This file resides in the `/templates/oauth` directory.

error.vm – Defines the presentation of error messages displayed to end users if there is a problem with the login or consent. This file resides in the `/templates/oauth` directory.

recover-password.vm – Defines the prompt for information to search for a user account so the password can be changed. This file resides in the `/templates/account` directory.

recover-password-verify.vm – Defines the prompt for the password change code sent by the Identity Broker and the new password. This file resides in the account directory of the `/templates` directory.

recover-password-success.vm – Defines the password change success notification. This file resides in the `/templates/account` directory.

recover-username.vm – Defines the prompt for information to search for an account username. This file resides in the `/templates/account` directory.

recover-username-verify.vm – Defines the prompt for the username recover code sent by the Identity Broker. This file resides in the `/templates/account` directory.

recover-username-success.vm – Displays the account username. This file resides in the `/templates/account` directory.

register.vm – Provides a form for creating a new user account. This file resides in the `/templates/account` directory.

register-success.vm – Defines the notification that the user account was successfully created. This file resides in the `/templates/account` directory.

If more than one template modification is needed, additional data can be added by adjusting or adding to the context objects that are present. See About Velocity Templates for general information about adding context. Use the following two consent objects when customizing these pages:

`$principal`

`$requestContext`

The $principal Object

The OAuth consent page header displays the currently logged in user's name as `$principal.username`. This is the placeholder variable used in the Velocity template.The Velocity template can be changed to display other attributes of the `$principal` object.

All of the SCIM principal attributes can be referenced by OAuth templates using SCIM's standard attribute notation. The following are examples for a SCIM object exposed as `principal` in the Velocity context.

Retrieving a simple attribute value:

```
$principal.userName
$principal.urn:scim:schemas:extension:enterprise:1.0:employeeNumber
```

Retrieving a complex attribute value:

```
$principal.name.givenName
$principal.urn:scim:schemas:extension:enterprise:1.0:manager.managerId
```

Retrieving multi-valued attribute values:

```
#foreach ( $email in $principal.emails )
$email.type: $email.value
#end
```

The $requestContext Object

This is the context placeholder for request-specific state, such as the current web application context, the current locale, or the current theme. The following are examples of `requestContext` in the Velocity context.

Retrieving the locale of the request:

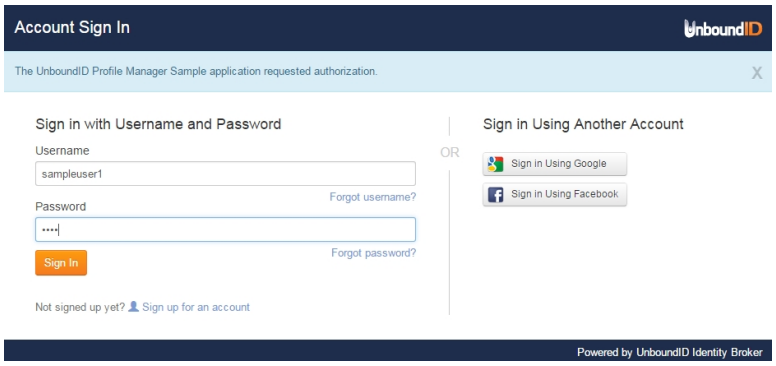
```
$requestContext.locale
```

Retrieving a Spring model object called 'token':

```
$requestContext.getModelObject('token')
```

User Account Registration and Recovery

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



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

Note

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

Options for account recovery and new account registration are enabled by configuring the OAuth HTTP Servlet Extension with the `dsconfig` tool on the Identity Broker server.

```
>>>> Configure the properties of the Oauth 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)   java-class                         com.unboundid.directory.broker.http.OAuth
                                         HTTPServletExtension
4)   response-header                     -
5)   register-enabled                    true
6)   register-dataview-name              User
7)   recover-username-enabled            false
8)   recover-username-scim-query         emails eq "$0" or phoneNumbers eq "$0"
9)   recover-username-validity-duration  5 m
10)  recover-username-full-text          Username Recovery Code: $0
11)  recover-username-compact-text       Username Recovery Code: $0
12)  recover-username-subject            Username Recovery Code
13)  recover-password-enabled            false
14)  recover-password-scim-query         userName eq "$0" or emails eq "$0" or phoneNumbe
rs eq "$0"
15)  recover-password-full-text          Password Change Code: $0
16)  recover-password-compact-text       Password Change Code: $0
17)  recover-password-subject            Password Change Code
18)  recaptcha-key                       reCAPTCHA will not be used
19)  recaptcha-secret                    reCAPTCHA will not be used
```

Note

Options for username recovery and password change are defined by the Identity Data Store password configuration. See the *UnboundID Identity Broker Installation Guide* for configuration details.

The option specific to user registration:

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

Options for user registration and username and password recovery:

register-dataview-name – Specifies the Data View in which the register self-service account flow creates new users, and the recover self-service account flows search for users.

recaptcha-key – Specifies the Google reCAPTCHA API key the register and recover self-service account flows should use. If a key is not specified, reCAPTCHA is not used.

recaptcha-secret – Specifies the Google reCAPTCHA API secret the register and recover self-service account flows should use. If a secret is not specified, reCAPTCHA will not be used by those flows.

Options specific to username and password recovery include:

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

If enabled, the Data Store should be configured with an OTP (one time password) Delivery Mechanism and a single-use-token Extended Operation Handler. See the *UnboundID Identity Broker Installation Guide* for more information.

recover-username-scim-query – Specifies the SCIM query used when the username recover self-service account flow searches for the account to recover.

recover-username-validity-duration – Specifies the duration the username recover code is valid before expiring.

recover-username-full-text – Specifies the full text sent with the username recover code, if the one time password mechanism supports long text.

recover-username-compact-text – Specifies the compact text sent with the username recover code, if the one time password mechanism does not support long text.

recover-username-subject – Specifies the subject sent with the username recover code when the one time password mechanism supports subjects.

recover-password-enabled – Specifies whether or not the password recover self-service account flow should be enabled. When disabled, the link does not display on the login page and any attempts to access the password recovery endpoint will result in a 403 Forbidden HTTP status code.

If enabled, the Data Store should be configured with an OTP Delivery Mechanism and a deliver-password-reset-token Extended Operation Handler. See the *UnboundID Identity Broker Installation Guide* for more information.

recover-password-scim-query – Specifies the SCIM query used when the password recovery self-service account flow searches for the account to recover.

recover-password-full-text – Specifies the full text sent with the password reset code, if the one time password mechanism supports long text.

recover-password-compact-text – Specifies the compact text sent with the password reset code, if the one time password mechanism does not support long text.

recover-password-subject – Specifies the subject sent with the password reset code when the one time password mechanism supports subjects.

Customizing the Identity Broker Application Logo

The Identity Broker's web application, can be changed or re-branded with a company logo. The application uses a cascading style sheet to determine appearance. The default style sheet file can be over written by creating a new style sheet for the Broker Console with the following naming convention:

`$HOME/.broker-console/branding-override.css`

If this file is present, the Identity Broker uses it to overwrite the existing style sheet.

The following is an example of the style sheet used to display the default logo in the title bar:

```
.product-logo {
 width: 18px;
 height: 24px;
 background-image: url("../img/unboundid-u30.png");
 background-size: 100% 100%;
}
```

Style changes take affect after the application is restarted.

To Customize the Logo

By default, the web applications look for the following branding override CSS file:

`~/.broker-console/branding-override.css`

where "~" is replaced for the home directory of the account the web server/application is running under. It is also possible to override this file name and location by setting the "`branding.override.file`" System Property. If this file is found, it is included after all of the other CSS files, so that it can override any of the application's styles.

The following is an example of the CSS used to display the default logo in the title bar:

```
.product-logo {
  width: 18px;
  height: 24px;
  background-image: url("../img/unboundid-u30.png");
  background-size: 100% 100%;
}
```

A `branding-override.css` file at `~/.broker-console` with the following contents will display a new logo after (restart the application after creating the file):

```
.product-logo {
  width: 550px;
  height: 190px;
  background-image: url(https://www.google.com/images/srpr/logo4w.png);
  background-size: 100% 100%;
}
```

Configuring Web Applications for Localization

To localize the Identity Broker web pages, create a set of resource bundles, for each language. Locale-specific data must be tailored according to the conventions of the language and region, and isolated into locale-specific objects in a Java `ResourceBundle`. The standard naming convention is `basename_<language1>_<country1>_<variant1>`. For example:

`messages_en_US.properties`

`messages_fr_FR.properties`

`messages_de.properties`

Each should have the same set of keys (for example `login_prompt`, `unknown_user`) and values, which are raw text in the appropriate language. Resource Bundles and internationalization for Java are described in `http://docs.oracle.com/javase/tutorial/i18n/resbundle/index.html`.

The resource bundles are loaded from the classpath as jar files in the `/lib` or `/lib/extensions` directories, or can be loaded as properties files in the server's `/classes` directory.

A Velocity Context Provider is then created to provide access to the resource bundles. Velocity Tools provide one that selects the appropriate bundle based on the locale determined from the incoming HTTP request and provides the messages from that bundle to the Velocity template. This tool class can be found at:

```
https://velocity.apache.org/tools/devel/javadoc/org/apache/velocity/tools/generic/ResourceTool.html
```

Configure an instance of this tool and specify the name of the resource bundle family ("messages" in this example). Create another properties file to configure the Velocity Tools classes:

```
/config/velocity/ResourceToolConfig.properties
```

Add the following lines:

`bundles=Messages`

`#locale=en_US` This can be used to enforce a specific locale.

Run the following `dsconfig` command to create and configure the Velocity Context Provider:

```
$ bin/dsconfig create-velocity-context-provider \
   --extension-name Velocity \
   --provider-name ResourceBundleProvider \
   --type velocity-tools \
   --set object-scope:session \
   --set included-view:/path/to/template \
   --set request
```

```
tool:org.apache.velocity.tools.generic.ResourceTool;config/velocity/ResourceTool.properti
es
```

The `included-view` is only necessary to make the localized messages available to only a certain set of templates.

About Velocity Templates

The Identity Broker exposes several Velocity pages through an HTTP Servlet Extension. The pages are for login, for OAuth consent, and an error page that can be surfaced for end users and are located in:

`<server-root>/config/velocity/templates`

See Configuring the Broker Login and Consent Pages for information about these files.

To enable Velocity support, add the Velocity HTTP Servlet Extension to an enabled HTTP or HTTPS connection handler:

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

Velocity template files contain presentation content and variables that are replaced when the content is requested. Variables are expressed using a `$` followed by an identifier that refers to an object put into a context (VelocityContext) by the server.

Velocity extensions can be configured to expose a number of objects in the context using the `expose-*` properties:

- **expose-request-attributes** – Indicates whether HTTP request attributes are accessible to templates using the `$ubid_request` variable. In general, request attributes are added by server components processing the HTTP request. Also the HTTP request parameters map is available as `$ubid_request.parameters`. Request parameters are supplied by the requester, usually in the request URL query string or in the body of the request itself.
- **expose-session-attributes** – Indicates whether HTTP session attributes are accessible to templates using the `$ubid_session` variable. Like request attributes, session attributes are also added by server components processing the HTTP request. The lifetime of these attributes persists until the user's session has ended.
- **expose-server-context** – Indicates whether a Server SDK server context is accessible to templates using the `$ubid_server` variable. The server context provides access to properties and additional information about the server. See the *Unbound ID Identity Broker Server SDK* documentation for more details.

The following are other properties of the Velocity HTTP Servlet Extension:

- **base-context-path** – URL base context for the Velocity Servlet.
- **static-content-directory** – In addition to templates, the Velocity Servlet will serve miscellaneous static content related to the templates. This property defines the directory where these resources are found.
- **static-context-path** – URL path beneath the base context where static content can be accessed.
- **mime-types-file** – Specifies a file that is used to map file extensions of static content to a Content Type to be returned with requests.
- **default-mime-type** – The default Content Type for HTTP responses. Additional content types are supported by defining on or more additional Velocity Template Loaders.
- **template-directory** – The directory where templates are stored. This directory also serves as a default for Template Loaders that do not have a template directory specified explicitly.

The VelocityContext object can be further customized by configuring additional Velocity context providers. The dot notation used for context references can be extended arbitrarily to access properties and methods of objects in context using Java Bean semantics. For example, if the HTTP request URL includes a `name` query string parameter like:

```
http://example.com:8080/view/hello?name=Karl&name=Vladimir+Ilyich&name=Steve
```

An HTML template like the following could be used to generate a page containing a friendly greeting to the end user:

```
<html>
  <body>
    Hello, $ubid_request.parameters.name[0],  $ubid_request.parameters.name[1], and
           $ubid_request.parameters.name[2]!
  </body>
</html>
```

Note

For security, all template substitutions are HTML escaped by default. To substitute unescaped content, a variable name ending with "WithHtml" must be used. For example, `$addressWithHtml`, would substitute the contents of the `$addressWithHtml` variable into the page generated from the HTML template without escaping it.

By default, the Velocity Servlet Extension expects to access content in subdirectories of the server's `config/velocity` directory:

- **templates** – This directory contains Velocity template files that are used to generate pages in response to client requests.
- **statics** – This directory contains static content such as CSS, HTML, and JavaScript files as well as images and third-party libraries.

Supporting Multiple Content Types

By default, the Velocity Servlet Extension is configured to respond to HTTP requests with a content type `text/html`. Change this request type by setting the default MIME type using `dsconfig`. For example, the following can be used to set the default type to XML:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name Velocity \
  --set default-mime-type:application/xml
```

HTML requests can be supported as well as clients that seek content in other formats. Create one or more Velocity Template Loaders to load templates for other content types like XML or JSON.

The ability to serve multiple formats of a document to clients at the same URL is typically called *content negotiation*. HTTP clients indicate the type of content desired using the `Accept` header. A client may use a header like the following to indicate that they prefer content in XML but will fallback to HTML if necessary:

```
Accept:  application/xml,text/html;q=0.9
```

The following can be used to create a Velocity Template Loader for XML content:

```
$ bin/dsconfig create-velocity-template-loader \
  --extension-name Velocity \
  --loader-name XML \
  --set evaluation-order-index:502 \
  --set mime-type-matcher:application/xml \
  --set mime-type:application/xml \
  --set template-suffix:.vm.xml
```

Upon receiving a request, the Velocity Servlet first creates an ordered list of requested media types from most desired to least based on the value of the `Accept` header. Starting from the most desired type, it will then iterate over the defined Template Loaders according to the `evaluation-order-index` property from lowest value to highest.

A Template Loader may indicate that it can handle content for requested media type by comparing the requested type to its `mime-type-matcher` property. A loader may be configured to load templates from a specific directory or load template files having a particular suffix. In this case, where XML templates are expected to be named using a `.vm.xml` suffix. If a loader indicates it handles the requested content type and a template exists for the requested view, the template is loaded and used to generate a response to the client. If no loaders are found for the requested media type, the next most preferred media type (if any) is tried. If no loaders indicated that they could satisfy the requested view, the client is sent an `HTTP 404 (not found)` error. If no loaders could provide acceptable media but the requested view exists in some other format, the client is sent an `HTTP 406 (not acceptable)` error.

In this example, a template file called `hello.vm.xml` can be used to generate a response in XML:

```
<hello name="$_request.parameters.name"/>
```

In this case, the response will contain an HTTP Content-Type header with the value of the `mime-type` property of the Velocity Template Loader.

Velocity Context Providers

The previous examples make use of value supplied as an HTTP request query string parameter to form a response. The templates contain a variable `$_request.parameters.name` that was replaced at runtime with a value from the Velocity Context.

The Velocity Extension can be configured to make some information available in the Velocity Context such as the HTTP request, session, and Server SDK Server Context. Velocity Context Providers provide more flexibility in populating the Velocity Context for template use.

Here are some of the properties of a Velocity Context Provider:

- **enabled** – Indicates whether the provider will contribute content for any requests.
- **object-scope** – Indicates to the provider how often objects contributed to the Velocity Context should be re-initialized. Possible values are: `request`, `session`, or `application`.
- **included-view/excluded-view** – These properties can be used to restrict the views for which a provider contributes content. A view name is the request URL's path to the resource without the Velocity Servlet's context or a leading forward slash. If one or more views are included, the provider will service requests for just the specified views. If one or more views are excluded, the provider will service requests for all but the excluded views.

Note

If the scope of the Velocity Tools context provider is constrained by setting the `included-view` property, the OAuth 2.0 consent flow may be affected. The `included-view` property should not be changed, unless all system default templates are included when setting the property.

Configuring HTTP Header Fields

By default, the Velocity Extension returns a set of standard HTTP header fields in every request served by the extension, including those for directing cache policies of user agents and frame-hosting options. These header fields can be configured in the following ways:

- The Velocity Servlet Extension's response-header configuration property can be specified to add a request header to every template request. The `static-response-header` property can be specified to add a header field to requests for static content like images and script files.
- Header fields for individual pages can be configured by using the `response-header` property of the Velocity Context Provider objects, which will add the header fields to just those pages served by the provider. Headers specified here will overwrite those specified by the Velocity Extension.
- Header fields can be manipulated directly by third-party Velocity Context Providers in code, adding or removing existing headers by manipulating the HTTP servlet response directly.

Handling Specific HTTP Methods in Third-Party Providers

In addition to contributing content to the Velocity Context, Velocity Context Providers can perform actions in response to particular HTTP methods. For example, a template can be used to POST a form of user data to a provider, which in turn would create a user in the User Store. In addition to handling HTTP GET and POST operations, a provider can handle any number of the standard HTTP methods (PUT, PATCH, DELETE, HEAD). Handling these methods is a two step process.

1. When creating a third-party Velocity Context Provider, configure the HTTP methods the provider will handle using the `http-method` property. For example, the following command might be used to configure a provider to handle GET and POST requests:

   ```
   $ dsconfig create-velocity-context-provider \
     --extension-name Velocity \
     --provider-name "My Provider" \
     --type third-party \
     --set http-method:GET \
     --set http-method:POST \
     --set extension-class:com.example.MyProvider
   ```

   Or update an existing provider:

   ```
   $ dsconfig set-velocity-context-provider-prop \
     --extension-name Velocity \
     --provider-name "My Provider" \
     --add http-method:POST
   ```

2. When implementing the Java class, override the `handlePost()` method adding code for handling POST operations. For example, use the internal `ServerContext` object to establish a connection to the server and create a new user using form data from a POST operation. Logic related to updating the context for the response may be implemented directly in the `handle<XXX>()` method or in the `updateContext()` method, which is called immediately after the relevant `handle<XXX>()` method.

Velocity Tools Context Provider

Apache's Velocity Tools project is focused on providing utility classes useful in template development. The Velocity Context can be configured by specifying Velocity Tool classes to be automatically added to the Velocity Context for template development. For more information about the Velocity Tools project, see `http://velocity.apache.org/tools/releases/2.0/`.

The following command can be used to list the set of Velocity Tools that are included in the Velocity Context for general use by templates:

```
$ bin/dsconfig get-velocity-context-provider-prop \
  --extension-name Velocity \
  --provider-name "Velocity Tools" \
  --property request-tool \
  --property session-tool \
  --property application-tool
```

Preserving Customized Files

Any files that are customized should be copied from the `config/velocity` subdirectories to the same subdirectory of the velocity directory under the server root (`<server-root>/velocity`). The files in `config/velocity` should not be modified. They are updated when the product is updated.

By default, any file of the same name under `<server-root>/velocity` will be loaded in place of `<server-root>/config/velocity`. This enables the preservation of customized files after a product upgrade.

After a product upgrade, review the files in `config/velocity` to determine if any changes should be incorporated into customized templates.

Addressing a Compromised Encryption Key

If an encryption settings definition becomes compromised, perform the following to create a new definition and update the Identity Broker servers. See the command line help for the `encryption-settings` tool for arguments.

Note

If the Identity Broker's encryption key is compromised, and the Broker has been collecting access tokens for external identity providers through the relying party feature, make sure those tokens are revoked.

1. Back up the encryption settings backend.
2. Back up the user store.
3. From the Identity Broker Console, revoke all authorizations for each application. See To Revoke All Authorizations.
4. Stop the HTTPS Connection Handler that is used for the Identity 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 Identity 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.

Managing the Log History Service

The Identity Broker provides an Authorization Log History Service that collects and indexes policy decision logs to support queries through the access history endpoint. The Broker uses an embedded implementation of Apache Lucene, an open-source Java-based indexing and search technology, to query through policy decision history.

About Multi-Broker Authorization Log Collection and Indexing

The default Identity Broker configuration enables the local indexing of policy decision records. The Authorization Log History Service is enabled on each Identity Broker when it is installed. For Identity Broker deployments that consist of a single Identity Broker server, the Authorization Log History Service can be used without modification.

Note

The Authorization Log History Service uses the Indexed Authorization Logger, which logs to `logs/.index-logs/authorization` by default, is intended for the Log History Service's exclusive use and should not be modified.

If an installation uses multiple Identity Brokers, the Log History Service should be configured to be hosted on one or more servers and to act as the data collectors. All other servers should have this service disabled, so that they can act as data sources.

The Authorization Log History service requires knowledge of all Identity Broker servers in an environment. Each indexing Identity Broker must poll all Identity Brokers in the set. An index with only part of the data is not supported. In a multi-copy configuration, each indexing Identity Broker maintains a complete copy of the index.

As the authorization log file is rotated on an Identity Broker, the Authorization Log History Service will poll for newly rotated files, collect them locally, and index them. Although the different copies may have different records at a single point in time (due to different polling cycles), they will ultimately be consistent by virtue of the required configuration.

Authorization log indexing requires very little CPU, but it does require a lot of disk space. When a set of Identity Brokers are working in a load balancing arrangement it may make sense to have more than one Identity Broker provide the indexing service. The primary reasons for a multi-copy configuration is to ensure high availability of the historical index and to protect against data loss due to disk failures.

About Index Latency

There is a latency between the time a record is added to an authorization log and the time the record shows up in the index. Log files are only indexed after they are rotated, and when the indexing Identity Broker is polling for new log files. The polling Identity Broker remembers the name of the last log file it indexed, and only accepts log files newer than the last indexed file. Newness is determined by the name of the rotated log file, so the indexing service is dependent on the rotated log file names to sort by the age of the file. The Authorization Log History Service has a configurable polling interval with a default of 5 minutes. The files are rotated every 30 minutes or after the file reaches 100 MB, whichever comes first. The latency is the larger of the Authorization Log History Service poll-interval and the average rotation period of the authorization log file.

Configuring Log Collection and Indexing

In a multi-Identity Broker configuration, on the server or servers that host the Authorization Log History Service, an Identity Broker External Server object must be created for each of the other servers in the deployment. A reference to each of these Identity Broker external servers must then be added to the host's Authorization Log History Service configuration.

Note

The Authorization Log History Service uses the Indexed Authorization Logger. When configuring properties, the properties for the Indexed Authorization Logger should be changed with caution. The `log-file` and `logged-message-type` properties should not be changed.

To Configure Log Collection and Indexing

The following is a sample configuration, where Server A and Server B host the Authorization Log History service, while Server C will be indexed by both Server A and Server B.

1. Create external servers on Server A that represent Server B and Server C.

```
$ dsconfig create-external-server --server-name ServerB \
  --type identity-broker \
  --set server-host-name:ServerB \
  --set "bind-dn:cn=directory manager" \
  --set password:****** \
  --set server-http-port:8443 \
  --set use-ssl:true
```

```
$ dsconfig create-external-server \
  --server-name ServerC \
  --type identity-broker \
  --set server-host-name:ServerC \
  --set "bind-dn:cn=directory manager" \
  --set password:****** \
  --set server-http-port:8443 \
  --set use-ssl:true
```

2. Configure Server A's authorization log history service to poll server B and server C.

```
$ dsconfig set-log-history-service-prop \
  --service-name Authorization \
  --set log-history-server:ServerB \
  --set log-history-server:ServerC
```

3. Create external servers on Server B that represent Server A and Server C.

```
$ dsconfig create-external-server \
  --server-name ServerA \
  --type identity-broker \
  --set server-host-name:ServerA \
  --set "bind-dn:cn=directory manager" \
  --set password:****** \
  --set server-http-port:8443 \
  --set use-ssl:true
```

```
$ dsconfig create-external-server \
  --server-name ServerC \
  --type identity-broker \
  --set server-host-name:ServerC \
  --set "bind-dn:cn=directory manager" \
  --set password:****** \
  --set server-http-port:8443 \
  --set use-ssl:true
```

4. Configure Server B's authorization log history service to poll server A and server C.

```
$ dsconfig set-log-history-service-prop \
  --service-name Authorization \
  --set log-history-server:ServerA \
  --set log-history-server:ServerC
```

5. Disable authorization log history service on Server C.

```
$ dsconfig set-log-history-service-prop \
  --service-name Authorization \
  --set enabled:false
```

At this point Server A and Server B will both be collecting log files from all three servers and indexing them, while Server C will not be indexing any log files.

About the Log History Service REST API Redirection

In a multi-Identity Broker configuration where one server is providing the Authorization Log History Service, all Identity Broker servers must implement the REST API that provides the information. This is handled automatically by the REST API. If an Identity Broker is not providing local indexing, the REST API returns an error indicating that it does not provide this service. After the indexing Identity Broker polls the other servers for an authorization log file, the non-indexing servers retain the indexing Identity Broker's information and redirect the REST API to that Identity Broker for all other transactions. If there is more than one indexing Identity Broker, the non-indexing broker will redirect to any one of indexing Identity Brokers.

The non-indexing Identity Broker caches the redirection target for five minutes and then updates the target during the next poll for a log file.

Index