UnboundID® Identity Broker

Application Developer Guide

Version 5.0.0

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 Application Developer Guide* provides information for client applications to interface with the UnboundID Identity Broker Server. We appreciate any feedback and requests for specific topics to cover in future revisions of this guide. Please send feedback to support@unboundid.com.

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 OneIdentity Platform. The OneIdentity 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 OneIdentity Platform provides a unified view of customer data across all applications, channels, partners, and lines of business.

The UnboundID OneIdentity 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 OneIdentity 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 software developers interested in developing applications that communicate with the Identity Broker API endpoints and request access to resources.

It is assumed that an installation of the Identity Broker Server exists and is accessible. Configuration must be performed and information must be gathered by the Identity Broker administrator to enable a client application to access the server. See What is Needed From the Identity Broker for more information.

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

- RESTful web services and principles.
- OAuth2 and OAuth2 Bearer Token specifications.
- OpenID Connect (OIDC).
- System for Cross-domain Identity Management (SCIM) protocol.
- Policy and attribute-based access control.

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

The UnboundID Identity Broker is an authorization and policy enforcement engine that securely exchanges customer data between applications and services. For companies managing large amounts of customer data, the Identity Broker serves as a gatekeeper of data access and automates the flow of customer data.

The Identity Broker Server powers OAuth 2.0, OpenID Connect, administration and policy services, each capable of handling millions of operations per day. The Identity Broker supports multiple REST API endpoints to enable client applications to access identity attributes.

This section explains Identity Broker features and components and includes the following:

Identity Broker Features

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

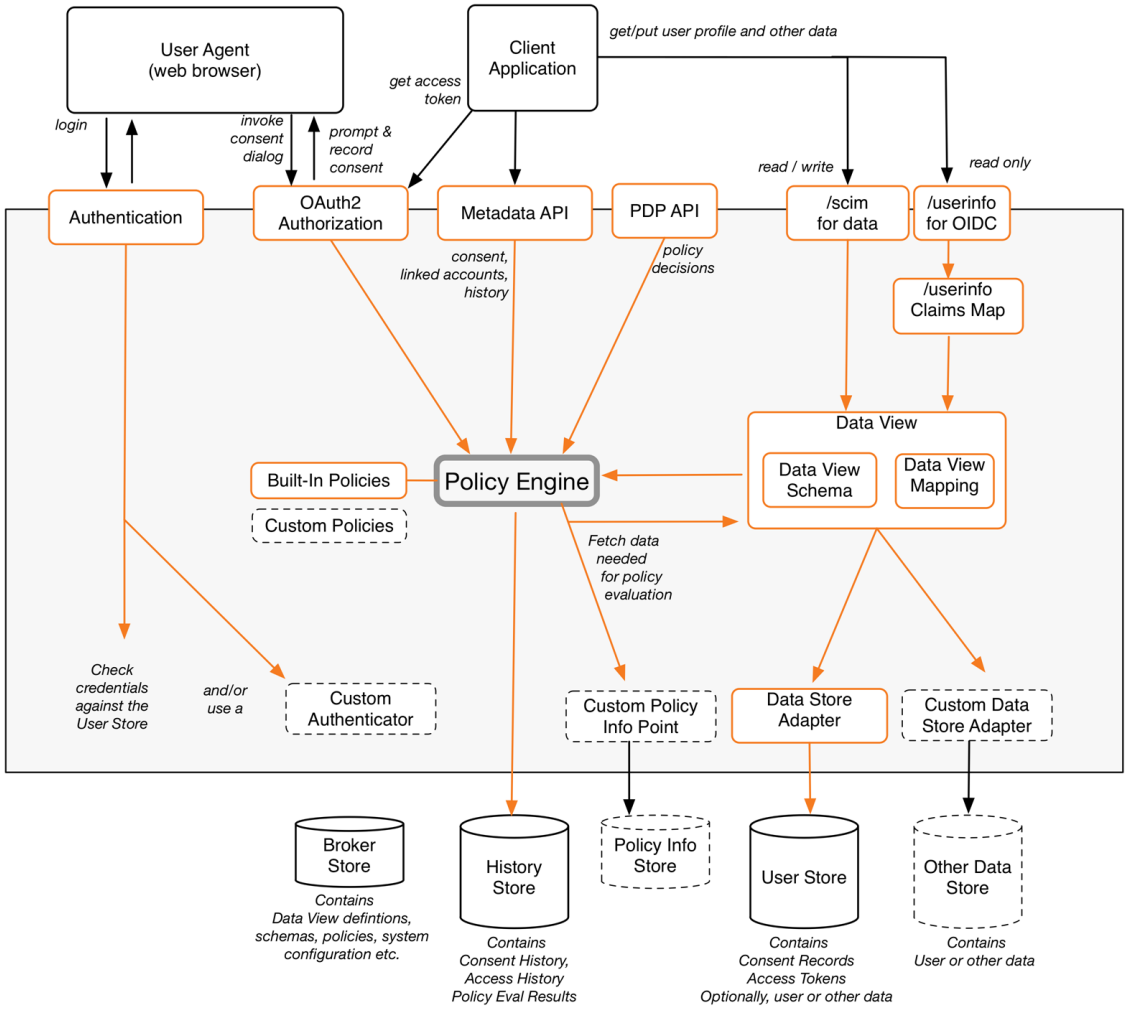
- **Support for multiple backend data stores**. The Identity Broker supports multiple data stores, with native support for the UnboundID Data Store and extension points for other data stores, such as relational databases. Applications can be written one time for access to the Identity Broker and receive 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 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. See Working with the Profile Manager Sample Application for information about managing end user consents.

Identity Broker Endpoints for Client Applications

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

Note

The Metadata APIs require a user ID. See Obtaining Usernames and UserIDs. If accessing records for the current authorized user, the parameter `self` can be used as the `<userID>`.

Identity Broker Endpoints for Client Applications

Endpoint	Description
/scim	
/scim/<name>	This is the SCIM protocol endpoint used to retrieve a specified data view, where `<name>` is the resource being accessed. This endpoint supports all SCIM operations and implements its access control through the Identity Broker's policies.
/oauth	
/oauth/authorize	The OAuth 2.0 standard authorization endpoint. This is the endpoint that an application will use to get an authorization grant from the user.
/oauth/token	The OAuth 2.0 token endpoint. This is the endpoint that an application will use to request an access token from the Identity Broker Server to access identity information.
/oauth/revoke	The Identity Broker endpoint used to revoke a token.
/oauth/validate	The Identity Broker endpoint used to validate a token.
/userinfo	
/userinfo	The OpenID Connect endpoint. Use this endpoint for applications that require read-only access to user profile data. Access to this endpoint requires an OAuth 2.0 access token with the `openid` scope. The client application will receive the attributes granted by the scopes in the access token. Either GET or POST actions can be used.
/metadata/v1/<userID>/accessHistory	
/<userID>/accessHistory	The Identity Broker endpoint used to retrieve a page of access history records that satisfy the provided query, page and sort parameters for the specified SCIM user ID. A request to this endpoint requires the `urn:unboundid:scope:read_access_history` scope.
/metadata/v1/<userID>/consentHistory	
/<userID>/consentHistory	The Identity Broker endpoint used to retrieve a page of consent history records that satisfy the provided query, page and sort parameters for the specified SCIM user ID. A request to this endpoint requires the `urn:unboundid:scope:read_consents` scope.
/metadata/v1//<userID>/consents	

Endpoint	Description
/<userID>/consents	The Identity Broker endpoint used to add, retrieve, or delete the consent granted by the specified SCIM user ID for application access to data. Either GET, POST, or DELETE actions can be used. A request to this endpoint requires either the `urn:unboundid:scope:read_consents` scope or the `urn:unboundid:scope:manage_consents` scope.
/<userID>/consents/applications	The Identity Broker endpoint used to retrieve the applications that have been granted consent by the specified SCIM user ID. A request to this endpoint requires the `urn:unboundid:scope:read_consents` scope.
/<userID>/consents/resources	The Identity Broker endpoint used to retrieve the resources to which the specified SCIM user ID has granted access. A request to this endpoint requires the `urn:unboundid:scope:read_consents` scope.
/metadata/v1/<userID>/links	
/<userID>/links	The Identity Broker endpoint used to add, retrieve, or delete the links to external identity provider accounts for the specified SCIM user ID. Either GET, POST, or DELETE actions can be used. A request to this endpoint requires either the `urn:unboundid:scope:read_links` scope or the `urn:unboundid:scope:manage_links` scope.
/<userID>/links/interactive	The Identity Broker endpoint used to initiate an interactive linking flow with an external identity provider. A request to this endpoint requires the `urn:unboundid:scope:manage_links` scope.

Chapter 2: Getting Started with Application Development

The Identity Broker Server provides two access endpoints for client applications to request end user resources:

UserInfo – The UserInfo endpoint (`/userinfo`) enables client applications to communicate with the Identity Broker to request access to claims (attributes) about the authenticated end user. The endpoint is read-only and cannot be used to update user data.

SCIM – The SCIM endpoint (`/scim/<name>`) enables client applications to connect with the Identity Broker to request access to end-user resources. Actions can be performed against the attributes if the Identity Broker policies allow.

Before designing an application to interact with the Identity Broker, determine the endpoint that the application will use for access and the settings that are in place (such as scopes and policies) that will affect the application's ability to access data.

This section describes what is required from the Identity Broker and includes the following:

What is Needed from the Identity Broker

Identity Broker configuration details will affect the client application's implementation and access to identity resources. The Identity Broker fully supports the role of Resource Server as defined within an OAuth2 context. Identity Broker configuration is performed through the Identity Broker Console interface or through the `broker-admin` command line tool. See the *UnboundID Identity Broker Administration Guide* for information about the console and Identity Broker configuration.

The Identity Broker administrator may have all of the configuration in place to enable access to a client application or may need specifics from the application developer. To develop client applications that can access the Identity Broker system, the following are required on the Identity Broker Server:

- **Register the application** – Registering an application with the Identity Broker defines the URL, the OAuth 2.0 grant types, token requirements, and the scopes that the application can use. A client ID and client secret are generated by the Identity Broker and are needed by the client application to interface with the `/oauth` endpoints. The Identity Broker administrator will need a redirect URL during the registration process so that the Identity Broker can redirect an end user back to the client application when authorizing access to resources. Self registration of an application can only be done through the Broker Admin APIs.
- **Define External Identity Providers** – If client applications are designed to enable user login through an external identity provider (Facebook, Google, or OpenID Connect), these providers must be configured for use through the Identity Broker. The Identity Broker must also be registered with the providers. See About the Identity Broker as Relying Party for details about the login and consent flow when external identity providers are enabled.
- **Define UserInfo Claims** – If using the UserInfo endpoint to access the Identity Broker, the client application will request the claims (identity resources) that the Identity Broker administrator has configured. Standard and custom claims are supported by the Identity Broker.
- **Define Scopes** – Scopes define the OpenID Connect scope and name that is displayed to end users of the client application, the claims that can be accessed, and the actions that can be performed. Scopes must be defined in the Identity Broker Server before a client application can include them in requests. Scopes are also used to capture consent for the requested resources. If custom scopes are needed by the client application, the Identity Broker administrator will need to create them.
- **Cross-origin Resource Sharing (CORS)** – Applications can make JavaScript calls to Identity Broker services that have CORS enabled. Trusted origins required by an

application can be specified when it is registered with the Identity Broker. HTTP Servlet Cross Origin Policies are defined for the servlets that will accept applications' JavaScript requests. See the *UnboundID Identity Broker Installation Guide* for details about HTTP Servlet Cross Origin Policies.

- **Customize Identity Broker Login and Consent pages** – The Identity Broker login and consent pages can be configured to display attributes of the client application. The pages are generated from Velocity templates located on the Identity Broker Server. Information about how to customize these templates is in the *UnboundID Identity Broker Administration Guide*.

OpenID Connect Scopes

OpenID Connect defines a set of standard scopes to determine which of the OpenID Connect claim values can be requested from the `/userinfo` endpoint. A set of standard scopes is installed with the Identity Broker. Additional or custom scopes can be created by the Identity Broker administrator.

In the Identity Broker, scopes are defined in terms of resources. Resource are generated from attributes defined in the SCIM Data View Schemas configured for the back-end data store. The OpenID Connect standard scopes are all predefined within the Identity Broker and reference the user attributes represented in the default User schema. For example, the resource `urn:scim:schemas:core:1.0:email` is defined by the OpenID Connect `email` scope.

OpenID Connect scopes and claims are documented in the specification (www.openid.net/specs). The only required scope is `openid`, which informs the Identity Broker that the client is making an OpenID Connect request. If the `openid` scope value is present, the Identity Broker will return an ID Token with an access token. The claims returned are governed by both Identity Broker policies and the scopes represented by the access token sent by the Identity Broker.

The scopes and claims available in the Identity Broker can be viewed in the Identity Broker Console or with the `broker-admin` command line tool. See the *UnboundID Identity Broker Administration Guide* for details.

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.

See the *UnboundID Identity Broker Administration Guide* for more information about policies.

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.

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.

Obtaining Usernames and User IDs

The Identity Broker default authentication scheme requires username and password credentials. To support additional authentication schemes, many of the Identity Broker REST APIs, such as the `/consents` API endpoint, require that end users be identified using a unique identifier rather than a username. This unique identifier is equivalent to a user's SCIM ID and can be obtained in the following ways:

- In the `user_id` field of an OAuth 2 token response.
- In the `user_id` field of an OAuth 2 token validation response.
- In the `sub` claim of a parsed OpenID Connect ID token.
- In the `sub` claim of an OpenID Connect UserInfo response.
- In the `urn:scim:schemas:core:1.0:id` value of a user's SCIM representation.

The Identity Broker REST API will accept `self` as a user ID to retrieve information for the owner of the OAuth 2.0 access token.

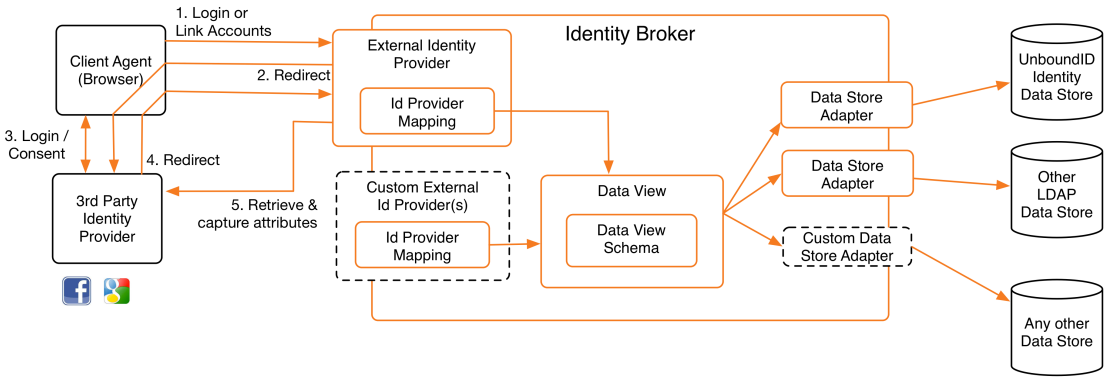
Character Length of Authorization Codes and Tokens

The authorization codes, access tokens, and refresh tokens issued by the Identity Broker are about 150 characters in length. This may be important for client applications persisting data.

Client IDs are standard universally unique identifiers (UUIDs) and are 36 characters.

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 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 the *UnboundID Identity Broker Administration Guide* 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. See Adding an Identity Provider Link to an Account for examples using the `/links` endpoint to link accounts.

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.

Working with the Sample Sign In 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.

The following are provided with the sample sign in application in `<server-root>/UnboundID-Broker/samples/sign-in.zip`:

- `README.txt` – describes how to configure and deploy the application either on the Identity Broker Server or on an external server.
- `sign-in.war` – the packaged web application that can be deployed on an external server. Included in this package are:
  - `ubid-broker-client.js` – a reusable script for the popup and redirect log in flows to the Identity Broker Server, and the UserInfo claims retrieval. This script uses OpenID Connect and the OAuth2 Implicit Grant authorization flow.
- `setup.sh`, `setup.bat` – the script to install the sample application on the Identity Broker Server.

Deploying the Sample Application

If the sample applications were not installed with the Identity Broker initial configuration, or if they need to be installed on a server other than the Identity Broker, perform the following steps to deploy the sample application:

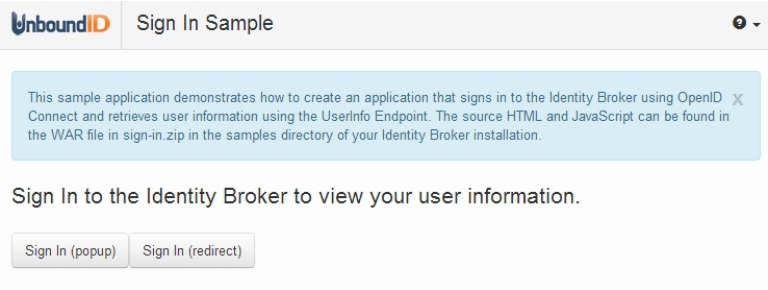
1. In the `<server-root>/UnboundID-Broker/samples` directory, unzip the `sign-in.zip` file.
2. Review the `README.txt` file for instructions on deploying the application within the Identity Broker Server or on an external server.
3. Launch the sample application in a browser with an address such as `https://<host:port>/samples/sign-in`.

Sign In Sample Application Pages

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

Landing Page

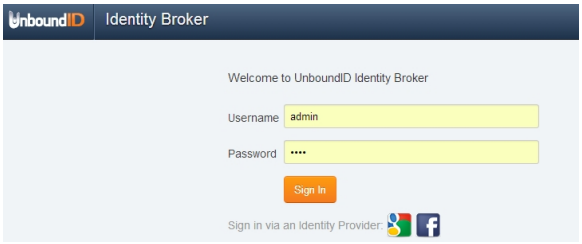
When the application is launched, the landing page displays.



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

Sign In Page

This is the Identity Broker login page, which can be configured from the Identity Broker Server. The end user enters account credentials into the fields. The account must exist in a data store that is configured to communicate with the Identity Broker Server. If the client application is configured to use an external identity provider to log in, an icon for that provider is displayed on the page. See About the Identity Broker as Relying Party for information about the login and account creation flows.



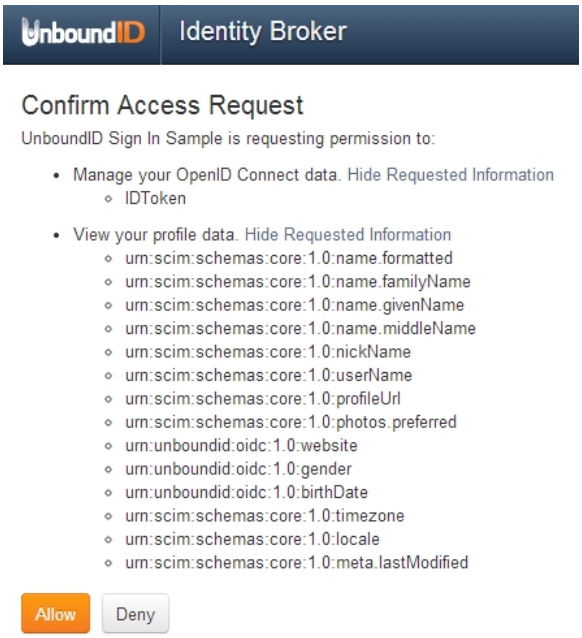
The application sends its client ID and a request to the Identity Broker for the attributes in the requested scopes. If no scope is provided, the Identity Broker will return the default values configured for the application.

Linked Accounts

If the application was configured to use an external identity provider as a login option, such as Google or Facebook, the identity provider and Identity Broker accounts can be linked. This requires the configuration of specific scopes. See the *UnboundID Identity Broker Administration Guide* for information.

Confirm Consent Page

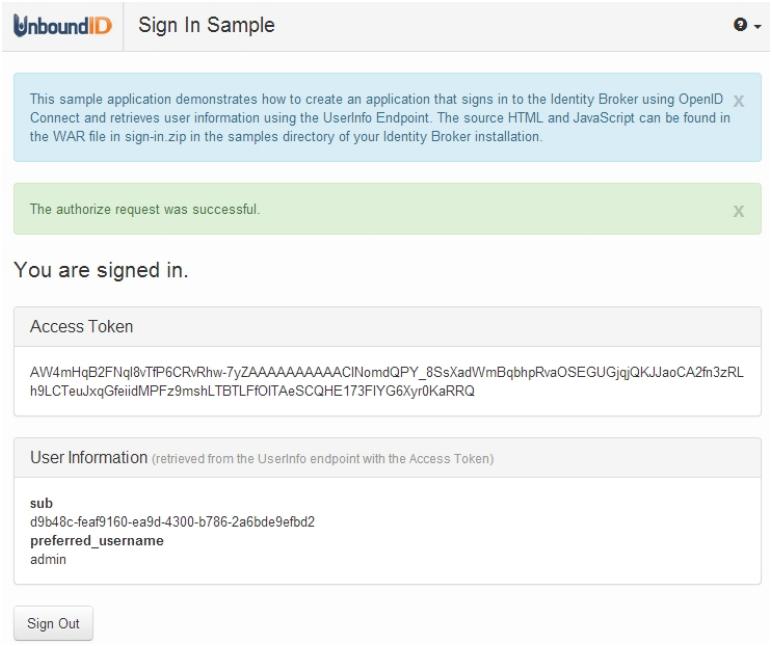
This is the Identity Broker consent page, which can be configured from the Identity Broker Server. The application returns a request for end user consent.



The end user can view the data requested from the profile by clicking the links on the page.

Approval Page

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



Sign Out

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

Working with the Profile Manager Application

The Profile Manager application displays how an end-users can view and manage the consents given to a client application that requested access to information. The consent and access history APIs used by this application are discussed in User Consent and Application Access Records.

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

- `README.txt` – describes how to configure and deploy the application either on the Identity Broker Server or on an external server.
- `profile-manager.war` – the packaged web application that can be deployed on an external server.
- `setup.sh`, `setup.bat` – the script to install the sample application on the Identity Broker Server, if it was not installed during the Identity Broker installation.

Deploying the Sample Application

If the sample applications were not installed with the Identity Broker initial configuration, or if they need to be installed on a server other than the Identity Broker, perform the following steps:

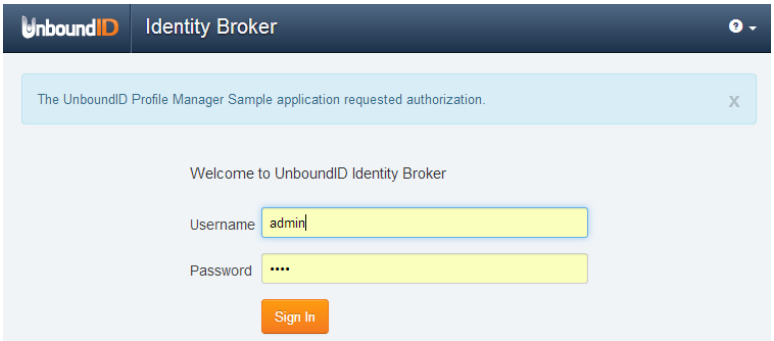
1. In the `<server-root>/UnboundID-Broker/samples` directory, unzip the `profile-manager.zip` file.
2. Review the `README.txt` file for instructions on deploying the application on an external server.
3. Launch the sample application in a browser with an address such as `https://<host:port>/samples/profile-manager`.

Profile Manager Application Pages

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

Landing Page

When the application is launched, the landing page displays.

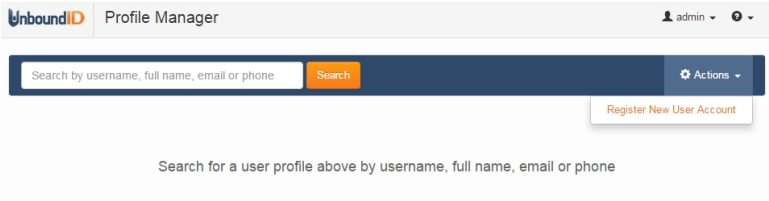


An end user can log into the Identity Broker. The account must exist in a data store that is configured to communicate with the Identity Broker Server. If the client application is configured to use an external identity provider to log in, an icon for that provider is displayed on the page. See About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

User Search Page

If logging into the application as the Identity Broker administrator, this page is displayed. End users will not see this page.

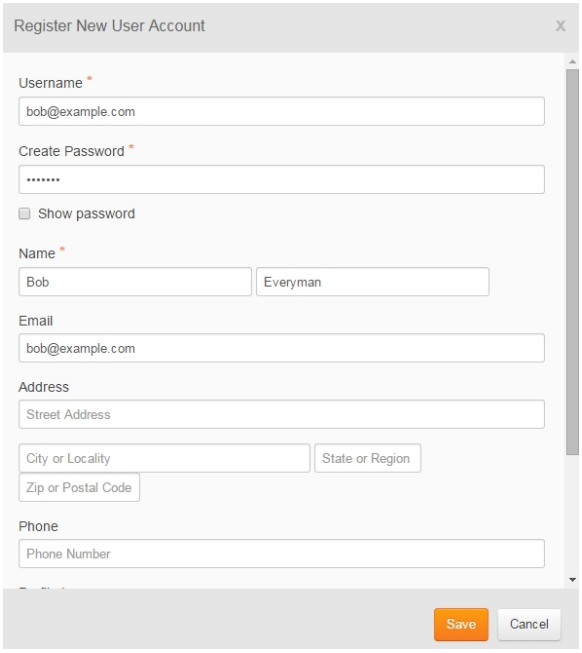
Enter a name, email address, or phone number to retrieve information for an end user. A new user account can also be created.



An existing user must reside in the backend user store that is configured for the Identity Broker, and that user store must be mapped to a Data View in the Identity Broker. If the Identity Broker was installed with sample data (an installation option), or if the `load-sample-data` tool was used post-install, two user accounts can be accessed: sampleuser1 and sampleuser2.

New User Registration

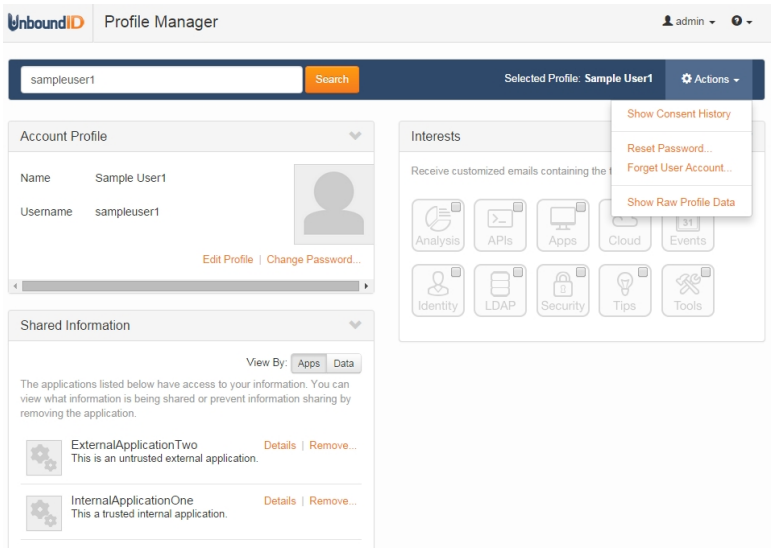
If registering a new user account, the following is displayed:



Enter the required information. The new account is added to the default User schema and the Users data view.

Profile Results Page

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



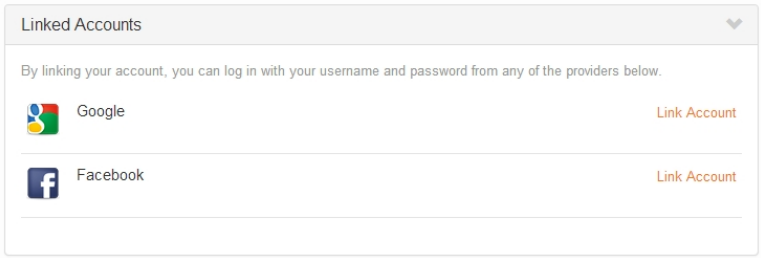
From this page, end users can perform the following:

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

- View and remove the applications that can access data.
- View and edit the types of information (Interests) that the user would like to see from an application.

Linked Accounts

If the application was configured to use an external identity provider as a login option, such as Google or Facebook, the identity provider and user accounts can be linked. This requires the configuration of specific scopes. See the *UnboundID Identity Broker Administration Guide* for information.



Chapter 3: Authentication

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

This chapter provides general information for applications to take the role of an OpenID Connect Relying Party while the Identity Broker acts as the OpenID Provider.

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

This section describes the OpenID Connect request and response flow through the Identity Broker and includes the following:

OpenID Connect Request

To authenticate an end user, a client application must have the following information from the Identity Broker Server administrator:

client identifier - An unique identifier issued to the client by the Identity Broker Server to identify itself.

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

authorization, token, validate, endpoint URLs - The Identity Broker's HTTP endpoint addresses for authenticating the end user, obtaining authorization, and issuing and validating access tokens. These are obtained from the Identity Broker administrator.

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

The client application uses this information to create an OAuth 2.0 request to obtain an access token.

The following example request uses the implicit grant flow:

```
GET /authorize?response_type=token%20id_token&client_id6c7283d2-92d6-4767-9ceb-ada61e5e7e
0d&state=4848573984983&scope=openid%20profile&
   redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

An OAuth 2.0 request becomes an OpenID Connect request with the inclusion of the `openid` scope. With the `openid` scope and the `response_type=id_token`, the client is requesting an identifier for the user as well as the ID token. The Identity Broker Policies will determine the attributes that the client application can access within any scopes that are defined.

OpenID Connect Response

If the end user logged in properly and authorized the client application request, the response from the Identity Broker Server includes an access token. If the request is an OpenID Connect request (contains the `openid` scope and `response_type=id_token`) the OAuth 2.0 access token response will include the `access_token` and `id_token` parameters. The following is encoded as a JSON Web Token in the `id_token`:

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

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

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

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

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

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

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

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

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

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

Chapter 4: Authorization Flows

The Identity Broker provides an OAuth 2.0, token-based authorization service that supports all OAuth 2.0 grant types outlined in RFC 6749. This service also provides additional functions to validate and revoke access tokens.

This section describes the different OAuth 2.0 authorization flows through the Identity Broker and includes the following:

About OAuth 2.0

The OAuth 2.0 authorization framework enables client applications to obtain access to protected resources by using tokens. The security and privacy of user information relies on the access requirements and consent flows configured for the client application. Consider the following when configuring an application to connect with the Identity Broker:

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

OAuth 2.0 Authorization Grant Types

The OAuth 2.0 specification states that a client application must receive authorization from a resource owner through an access token, to retrieve the owner's protected resources. The Identity Broker supports all OAuth 2.0 authorization grant types:

- **Authorization Code Grant** – This is a server-side redirection-based flow. The client application redirects the end user (user agent) to the authorization endpoint (Identity Broker) to grant or deny access to a resource. If access is granted, the Identity Broker returns a redirection URI with an authorization code and then redirects the end user back to the client application. The client application uses the authorization code to request an access token from the Identity Broker Server. The Identity Broker validates the authorization code and returns an access or refresh token to the client. The client application can now use the access token to request resources. The access token serves as both authentication of the client, and authorization to access the resources.
- **Implicit Code Grant** – This is another redirection-flow, designed for web applications, such as mobile applications or JavaScript applications running in browsers. The flow is

similar to the authorization grant flow, except that the Identity Broker redirects the client application with an embedded access token in the URI, rather than an authorization code requiring a separate token request. The client secret is not used because it would be stored (and be vulnerable) in the client. No refresh token is sent as this grant type is designed for short-lived access to a resource.

- **Resource Owner Password Credentials Grant** – This flow enables a user to log in with a username and password to receive an access token. The client application can then keep the access token for access to resources. The client is expected to discard the username and password and keep the access token. This flow should only be used for trusted client applications.
- **Client Credentials Grant** – This flow enables a client's application server to exchange the client ID and the client secret for an access token. This enables applications to directly access resources that are specific to the application and are not tied to an identity.

Issuing Authorization Code Grant Requests

The Authorization Code Grant Flow follows these basic steps:

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

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

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

Example Redirection

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

Step 2. Resource Owner Authenticates and Grants Authorization

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

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

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

Example Authorization Header Response

```
HTTP/1.1 302 Found
Location: https://<server2.example.com>?code=MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-vzK
YVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKm4QVx6mCTmT9gztIn45K9KKJ22p8IiJHiLXGEg2oUV&state=
48389488
```

Step 4. Exchange Authorization Code for an Access Token

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

Example Request

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

grant_type=authorization_code&code=MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-vzKYVltlykXrZ
E2AG0F3J3mQjUYOSP3dCOaIeYEUWSKnav_aXvvyuxT3ogtZT-dgNZEnk6X0XaoPf6BVlVRibA
&redirect_uri=https%3A%2F%2Fserver2%2Eexample%2Ecom
```

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

Example Response

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

{
  "access_token":"MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-vzKYVltlykXrZE2AG0F3J3mQjUYOSP
3dCOaIeYEUWSKMYeiJy-24paR9YLEZpKDc-mwlE4ML8LRqAyhPMtAoBA",
  "token_type":"bearer",
  "expires_in":41558,
  "scope":"email profile"
}
```

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

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

Example Request

```
GET /scim/resource HTTP/1.1
Host: server.example.com
Authorization: Bearer MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-vzKYVltlykXrZE2AG0F3J3mQjU
YOSP3dCOaIeYEUWSKMYeiJy-24paR9YLEZpKDc-mwlE4ML8LRqAyhPMtAoBA
```

The resource server returns the requested information.

Issuing Implicit Code Grant Requests

The Implicit Code Grant Flow follows these basic steps:

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

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

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

Example Redirection

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

Step 2. Resource Owner Authenticates and Grants Authorization

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

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

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

Example Redirect Response

```
HTTP/1.1 302 Found
Location: https://<server2.example.com>/callback#access_token=1MF2AAQGBBlpxSGUtUYJQo2oB1p
1kw3CNcM5QRmok-vzKYVltlykXrZE2AG0F3J3mQjUYOSP3dCOaIeYEUWSKMYeiJy-24paR9YLEZpKDc-mwlE4ML8L
RqAyhPMtAoBA&token_type=bearer&state=4848573984983&expires_in=43062
```

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

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

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

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

Example Request

```
GET /scim/resource HTTP/1.1
Host: <server.example.com>
Authorization: Bearer MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-vzKYVltlykXrZE2AG0F3J3mQjU
YOSP3dCOaIeYEUWSKMYeiJy-24paR9YLEZpKDc-mwlE4ML8LRqAyhPMtAoBA
```

The resource server returns the requested information.

Issuing Resource Owner Password Credentials Requests

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

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

Step 1. Client Asks for Resource Owner's Credentials

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

Step 2. Client Makes an Authorization Request at Token Endpoint

The client makes an authorization request to the Identity Broker's token endpoint by passing in the `client_id` and `client_secret` and the resource owner's username and password. The `client_id` and `client_secret` can be passed on in two ways: as a basic authentication request header or as part of the parameters passed in the body of the request.

Example Authorization Header Request

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

```
Authorization: Basic <Base64-encoded client_id:client_secret>
```

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

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

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

Example Authorization Response

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

{
  "access_token":"MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-vzKYVltlykXrZE2AG0F3J3mQjUYOSP
3dCOaIeYEUWSKFEDrIpaEn5N9MfAm1BjZ5OYLHu0L823L2JsMn7i2wug",
  "token_type":"bearer",
  "expires_in":42203,
  "scope":"profile",
}
```

Issuing Client Credentials Code Requests

The client credentials grant flow follows these basic steps:

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

Step 1. Client Makes an Authorization Request at Token Endpoint

The client makes an authorization request to the Identity Broker's Token endpoint by passing the `client_id` and `client_secret` . The `client_id` and `client_secret` can be passed on in

two ways: as a basic authentication request header or as part of the parameters passed in the body of the request.

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

```
Authorization: Basic <Base64-encoded client_id:client_secret>
```

Example Authorization Header Request

```
POST /oauth/token?grant_type=client_credentials
Host: server.example.com
Authorization: Basic czQER9k3dD94aIdplr957Udk8
Content-Type: application/w-www-form-urlencoded
```

Step 2. Client Receives the Access Token

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

Example Access Token Response

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

{
  "access_token":"MF2AAQGBBlpxSGUtUYJQo2oB1p1kw3CNcM5QRmok-vzKYVltlykXrZE2AG0F3J3mQjUYOSP
3dCOaIeYEUWSKFEDrIpaEn5N9MfAm1BjZ5OYLHu0L823L2JsMn7i2wug",
  "token_type":"bearer",
  "expires_in":42203,
  "scope":"profile",
}
```

The Identity Broker Token Endpoint

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

Request

The following example makes a token request to the endpoint:

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

```
grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA&redirect_uri=https%3A%2F%2Fclie
nt%2Eexample%2Ecom%2Fcb
```

Response

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

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

{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "bearer",
    "expires_in": 3600,
    "scope": "openid email profile",
    "scope_info": {
       "email": {
           "action": "Read",
           "purpose": "Any",
           "resources": [
               "urn:scim:schemas:core:1.0:emails.preferred",
               "urn:unboundid:oidc:1.0:emailVerified"
          ]
       },
       "openid": {
          "action": "Any",
          "purpose": "Any",
          "resources": [
              "urn:unboundid:resources:broker:IDToken"
           ]
        },
        "profile": {
          "action": "Read",
          "purpose": "Any",
          "resources": [
              "urn:scim:schemas:core:1.0:name.formatted",
              "urn:scim:schemas:core:1.0:name.familyName",
              "urn:scim:schemas:core:1.0:name.givenName",
              "urn:scim:schemas:core:1.0:name.middleName",
              "urn:scim:schemas:core:1.0:nickName",
              "urn:scim:schemas:core:1.0:userName",
              "urn:scim:schemas:core:1.0:profileUrl",
              "urn:scim:schemas:core:1.0:photos.preferred",
              "urn:unboundid:oidc:1.0:birthDate",
              "urn:scim:schemas:core:1.0:timezone",
              "urn:scim:schemas:core:1.0:locale",
              "urn:scim:schemas:core:1.0:meta.lastModified"
            ]
         }
     },
      "user_id": "9f8a23-cccc76ee-d07b-3b8c-922c-ddd809c4c173",
      "id_token": "eyJhbGciOiJIUzI1NiJ9.eyJhdXRoX3RpbWUiOjE0MjE4ODExMDMsImV4
                  cCI6MTQyMTg4MjAwOSwic3ViIjoiOWY4YTIzLWNjY2M3NmVlLWQwN2ItM2I
                  4Yy05MjJjLWRkZDgwOWM0YzE3MyIsImF1ZCI6WyJhY21lIl0sImlzcyI6Im
```

```
                  h0dHBzOlwvXC94MjI1MC0wMS5leGFtcGxlLmNvbSIsImlhdCI6MTQyMTg4M
                  TEwOX0.CZYpxocXZ-_DEPttmHqSiQ1FU8Pplb8I-7oK3PMp4-Y"
}
```

Token Validation by the Identity Broker

The Identity Broker token validation endpoint (`/oauth/validate`) uses pre-shared client credentials to validate access tokens. To validate an access token, a POST is sent to the Identity Broker's `/oauth/validate` endpoint, which returns a response with additional information about the resource owner and scopes.

Parameters can be provided as query parameters appended to the token validation endpoint URL. The `access_token` parameter is required. The `id_token` parameter is optional. If both are provided, the validation endpoint verifies that the ID token was issued with the access token.

An application can validate an ID token itself, if designed to do so. Refer to the OpenID Connect Core 1.0 specification for information. If a `nonce` value was provided during an implicit OpenID request flow, an ID token validation response should include the same `nonce` value. The client application should make sure that the values match.

If a `client_id` value is provided, it must belong to the same application that was used to request the `access_token`.

Request

The following is a request to validate a token:

```
POST /oauth/validate?token=<access token>&id_token=<id token>
Host: example.com
Accept: application/json
```

Response

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

```
response:
{
  "user_ID":"scim_userID",
  "scope_info": {
      "profile": {
         resource: [<resource_urns>],
         action: <action>,
         purpose: <purpose>
  },
 "nonce":"165297",
 "user_id":"d9b48c-31c06853-13e3-4aea-841f-bdc0b18b300d",
 "client_id":"@sample-sign-in@",
 "issued_at":"20140514153805Z",
 "expires_in":43200,
 "auth_time":"20140514153804Z",
 "id_token_issued_at":"20140514153805Z"
}
```

If validation fails for any reason, an HTTP 400 status code is returned.

Token Revocation by the Identity Broker

The token revocation endpoint (`/oauth/revoke`) enables clients to send a POST request to the Identity Broker to revoke access or refresh tokens. This may be used when the client logs out of or uninstalls the application. Revoking a token does not remove any associated consents.

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

For example, he following revokes a token:

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

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

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

Obtaining a Refresh Token

To request an OAuth 2.0 refresh token, either the `offline_access` or `urn:unboundid:scope:refresh_token` scope should be requested in the client application's authorization request. The client application's use and consent requirements will dictate the choice of scope:

- The `offline_access` scope is provided for compliance with the OpenID Connect specification. To successfully obtain a refresh token, a client using this scope must also specify the prompt authorization request parameter with a value of `consent`. End users must provide explicit consent to grant a refresh token every time one is requested.
- The `urn:unboundid:scope:refresh_token` does not require the use of the prompt authorization parameter.

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

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

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

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

The response will look like the following:

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

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

Chapter 5: Accessing Data

The Identity Broker server supports two user profile endpoints:

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

Access to resources is determined by the policies that are configured on the Identity Broker Server. If an application request to the Identity Broker is delivering partial results, it may be due to policy settings. See How Policy Affects the Data Returned to an Application.

This section describes data access from the Identity Broker and includes the following:

The Data View (SCIM) Endpoint

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

No Support for HTTP PUT. The SCIM endpoint does not support the HTTP PUT operation, because PUT assumes that the client has access to all the attributes. The client application may not have access to some attributes based on policies or consents.

No Sorting. The Data View endpoint does not support sorting search results.

Self Resource. The Identity Broker supports a special resource type, `Self`, to retrieve attributes of the currently authenticated user without knowing the SCIM ID. Retrieve attributes with the SCIM ID `Self` with the following:

```
/scim/Self/Self
```

Or retrieve the profile using the list/query method, which always returns one resource:

```
/scim/Self
```

Authentication. The SCIM endpoints are protected by bearer token authentication, obtained from the Identity Broker. See Authentication for details.

The following table describes SCIM features and whether they are supported by the Identity Broker.

SCIM Feature	Description
JSON	Yes
XML*	Yes
Authentication/Authorization	Yes
Service Provider Configuration	Yes
Schema endpoint	Yes
Resource retrieval via GET	Yes
List/query resources	Yes
Query filtering*	Yes
Query result sorting*	No
Query result pagination*	Yes
Resource updates via PUT	No
Partial resource updates via PATCH*	Yes
Resource deletes via DELETE	Yes
Resource versioning*	No
Bulk*	Yes
HTTP method overloading	Yes

* Denotes an optional feature of the SCIM Protocol.

Data View Examples

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

GET (Data View Schemas)

The following is an example call to the Identity Broker `/scim/Schemas/{name}` endpoint to get the Identity Broker schema `User`. If a `{name}` is not specified, all Identity Broker schemas are returned.

Request

```
GET /scim/Schemas/User
Host: example.com
Accept: application/json
Authorization: Bearer MF2AAQGBBlY1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-
LHJGSkZwAd3dYWPVERzIAy-VczegSxSR2c51uoiFgSyQFfC_y0kLy15L4iTI
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: …
{
  "name":"User",
  "description":"…",
  "schema":"urn:unboundid:schemas:broker:1.0",
  "endpoint":"/Users",
  "id":"urn:unboundid:schemas:broker:1.0:User",
  "meta":{ "location":"https://<example.com>:8445/scim/v1/Schemas/urn:unboundid:schemas:b
roker:1.0:User"
  },
  "attributes":[
    {
      "name":"displayName",
      "type":"string",
      "multiValued":false,
      "description":"The name of the User, suitable for display to end-users.",
      "schema":"urn:scim:schemas:core:1.0",
      "readOnly":false,
      "required":false,
      "caseExact":false
    },
    ... // other attributes
  ]
}
```

jQuery Example

```
$.ajax({
type: "GET",
url: "https://example.com/scim/Schemas/User",
headers: { "Authorization": "Bearer " + accessToken },
dataType: "json",
success: function(schemas) {
 }
});
```

GET

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

Request

```
GET /scim/Users?startIndex=1&count=10&filter=userName+sw+%22sam%22
Host: example.com
Accept: application/json
Authorization: Bearer MF2AAQGBBlY1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-
LHJGSkZwAd3dYWPVERzIAy-VczegSxSR2c51uoiFgSyQFfC_y0kLy15L4iTI
```

Response

The data returned is dependent on the Identity Broker configuration and the Policies in place.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: …
{
  "totalResults":1,
  "itemsPerPage":1,
  "startIndex":1,
  "schemas":[
    "urn:unboundid:oidc:1.0",
    "urn:scim:schemas:core:1.0",
    "urn:unboundid:profile:1.0"
  ],
  "Resources":[
    {
      "name":{
        "givenName":"Sample",
        "familyName":"User1",
        "formatted":"Sample User1"
       },
       ...// other user properties
     },
    ...// other users
  ]
}
```

jQuery Example

```
$.ajax({
 type: "GET",
 url: "https://example.com/scim/Users",
 data: { startIndex: 1, count: 10, filter: 'userName sw "sam"'},
 headers: { "Authorization": "Bearer " + accessToken },
 dataType: "json",
 success: function(usersPage) {
// application can do something with returned data...
 }
});
```

GET (by User ID)

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

Request

```
GET /scim/Users/9f8a23-47c7be45-0ce5-3105-8ea8-fc3c39c47f91
Host: example.com
Accept: application/json
Authorization: Bearer MF2AAQGBBlY1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-
LHJGSkZwAd3dYWPVERzIAy-VczegSxSR2c51uoiFgSyQFfC_y0kLy15L4iTI
```

Response

The data returned is dependent on the Identity Broker configuration and the Policies in place.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: …
{
  "schemas":[
    "urn:unboundid:oidc:1.0",
    "urn:scim:schemas:core:1.0",
    "urn:unboundid:profile:1.0"
  ],
  "name":{
    "givenName":"Sample",
    "familyName":"User1",
    "formatted":"Sample User1"
  },
  ... // other user properties
}
```

jQuery

```
$.ajax({
 type: "GET",
 url: "https://example.com/scim/Users/",+userId,
 data: { startIndex: 1, count: 10, filter: 'userName sw "sam"'},
 headers: { "Authorization": "Bearer " + accessToken },
 dataType: "json",
 success: function(user) {
//  application can do something with returned data...
 }
});
```

POST

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

Request

```
POST /scim/Users
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer MF2AAQGBBlY1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-
LHJGSkZwAd3dYWPVERzIAy-VczegSxSR2c51uoiFgSyQFfC_y0kLy15L4iTI
Content-Length: ...
{
 "schemas": [ "urn:unboundid:oidc:1.0", "urn:scim:schemas:core:1.0" ],
 "name": {
 "formatted": "Another Sample User III",
 "familyName":"User",
 "givenName":"Another",
 "middleName":"Sample"
},
 "userName":"sampleuser3"
}
```

Response

The data returned is dependent on the Identity Broker configuration and the Policies in place.

```
HTTP/1.1 201
Created Content-Type: application/json
Content-Length: …
{
  "schemas":[
    "urn:unboundid:oidc:1.0",
    "urn:scim:schemas:core:1.0",
    "urn:unboundid:profile:1.0"
  ],
  "name":{
    "givenName":"Another",
    "familyName":"User",
```

```
    "formatted":"Another Sample User III"
  },
  "id":"9f8a23-3562ddf5-50d0-4aac-a761-7ecb9bcb7633",
  "userName":"sampleuser3",
  "meta":{
    "created":"2014-09-04T19:06:22.547Z",
    "lastModified":"2014-09-04T19:06:22.547Z",
    "location":"https://example.com/scim/v1/Users/9f8a23-3562ddf5-50d0-4aac-a761-7ecb9bcb
7633"
  }
}
```

jQuery Example

```
$.ajax({
 type: "POST",
 url: "https://example.com/scim/Users",
 data: JSON.stringify({
   "schemas": [ "urn:unboundid:oidc:1.0", "urn:scim:schemas:core:1.0" ],
   "name": {
   "formatted": "Another Sample User III",
   "familyName":"User",
   "givenName":"Another",
   "middleName":"Sample"
 },
  "userName":"sampleuser3"
 }),
 headers: { "Authorization": "Bearer " + accessToken },
 contentType: "application/json"
 dataType: "json",
 success: function(user) {
// returned data sample...
 }
});
```

Note

Creating a user through SCIM is governed by Identity Broker Policy. The Identity Broker administrator will need to provide specifics about what this Policy will allow.

UPDATE

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

Request

```
PATCH /scim/Users/9f8a23-31c5b68d-2c8d-4dd2-987b-09627cb1ff2d
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer MF2AAQGBBlY1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-
LHJGSkZwAd3dYWPVERzIAy-VczegSxSR2c51uoiFgSyQFfC_y0kLy15L4iTI
Content-Length: ...
```

```
{
 "schemas": [ "urn:unboundid:oidc:1.0", "urn:scim:schemas:core:1.0" ],
 "name": {
 "formatted": "My Sample Tester III",
 "familyName":"Tester",
 "givenName":"My",
 "middleName":"Sample"
 }
}
```

Response

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

jQuery Example

```
$.ajax({
 type: "PATCH",
 url: "https://example.com/scim/Users/"+userId,
 data: JSON.stringify({
   "schemas": [ "urn:unboundid:oidc:1.0", "urn:scim:schemas:core:1.0" ],
   "name": {
   "formatted": "My Sample Tester III",
   "familyName":"Tester",
   "givenName":"My",
   "middleName":"Sample"
  }
 }),
 headers: { "Authorization": "Bearer " + accessToken },
 contentType: "application/json",
 success: function(){
// no data returned...
 }
});
```

DELETE

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

Request

```
DELETE /scim/Users/9f8a23-47c7be45-0ce5-3105-8ea8-fc3c39c47f91
Host: example.com
Authorization: Bearer MF2AAQGBBlY1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-
LHJGSkZwAd3dYWPVERzIAy-VczegSxSR2c51uoiFgSyQFfC_y0kLy15L4iTI
/9f8a23-47c7be45-0ce5-3105-8ea8-fc3c39c47f91==the user's ID
```

Response

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

jQuery Example

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

// no data returned...

 }
});
```

UserInfo Access Example

A client application accesses the `/userinfo` endpoint by passing an HTTP `GET` request with an access token parameter to the Identity Broker Server. The response is a JSON object.

Request

The following is a Java Script example call to the Identity Broker `/userinfo` end point:

```
GET /userinfo
Host: <example.com>
Accept: application/json
Authorization: Bearer MF2AAQGBBlY1UzNKUYJQgOqihaEJvCvPok4pYLR0a-9XOHkWCQqJ9wCHB66kwESoaO-
LHJGSkZwAd3dYWPVERzIAy-VczegSxSR2c51uoiFgSyQFfC_y0kLy15L4iTI
```

Response

The data returned is dependent on the Identity Broker configuration and the Policies in place.

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

jQuery Example

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

The Identity Broker Logout Endpoint

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

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

If a `redirect_uri` is not provided, the browser will be redirected to the configured `default-logout-success-url` for the Spring Security HTTP Servlet Extension (which defaults to `/view/login`). If a `redirect_uri` is provided, then `client_id` must also be provided. The `redirect_uri` value must match one of the redirect URIs configured for the application (which is retrieved by the `client_id`). The browser will be redirected to the provided `redirect_uri` after logout.

Request

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

```
POST /logout.do?client_id=385b45d0-88bd-4973-a9bc-06484ad27e42&redirect_uri=https://examp
le-app.com/
Host: example.com
Content-Length: 0
Cookie: JSESSIONID=xpdpr7z6fxh31rjdpygcmce0c
```

Response

The following is an example response:

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

User Metadata

An application can provide consent management to end users through a series of Metadata APIs. These are all illustrated by the Profile Manager sample application. These APIs rely on scopes and resources, and must pass through the Identity Broker policy engine to access data.

Note

The scopes that are listed in this section are those that were installed with the Identity Broker. They can be changed or new scopes can be added to tailor access to data. Review the defined scopes and policy requirements with the Identity Broker administrator.

For each endpoint, a value of `self` can be used for the `<userID>`variable. This will retrieve data for the currently authenticated owner of the access token.

Managing Access History Records

Data access history can be retrieved for an end user by calling the `/metadata/v1/<userID>/accessHistory` endpoint. The Identity Broker installs the following scope to retrieve access history records:

`read_access_history` – Enables reading the access history records for the specified user ID, and includes the following resource:

```
urn:unboundid:resources:broker_metadata:accessHistory
```

Read Access History Examples

Request:

```
GET /metadata/v1/9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b/accessHistory?application=My
App&decision=PERMIT&sortBy=timestamp&sortOrder=descending&startIndex=0&count=1
Host: <example.com>
Accept: application/json
Authorization: Bearer Aes-6SPszrDDpFxKuCdDqDxoZSdqAAAAAAAAAAB-sedGtKSBOaJdg3opJsRtLyqqF_k
uE92iiVFvi0LIqXYcjrqQK-6HVhqGUyWiDP84kpmZaMm9pestt4O2PVyVlWrd__6wa4rU_NLVelrleA
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
{
  "startIndex":0,
  "count":10,
  "totalResults":45,
  "data":[
    {
      "action": {
         "name":"Read",
         ... // other action properties
    },
      "application": {
         "name":"MyApp",
         ... // other application properties
    },
```

```
      "appliedPolicies": [
        "urn:unboundid:policy:TrustLevelPolicy",
        "urn:unboundid:policy:GovernanceTagPolicy",
        "urn:unboundid:policy:Basic Consent"
    ],
    "decision":"PERMIT",
    "owner":"9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b",
    "purpose": {
       "name":"Any",
       ... // other purpose properties
    },
    "resources": [
        {
          "urn":"urn:scim:schemas:core:1.0:name.formatted",
          ... // other resource properties
      },
        ... // other resources
    ],
      "timestamp":1409779918000
  },
    ... // other data entries
 ]
}
```

jQuery Example:

```
$.ajax({
  type: "GET",
  url: "https://<example.com>/metadata/v1/" + userId + "/accessHistory?application=MyApp&
decision=PERMIT&sortBy=timestamp&sortOrder=descending&startIndex=0&count=10",
  headers: { "Authorization": "Bearer " + accessToken },
  dataType: "json",
  success: function(data) {
    // do something interesting with the returned history records
  }
});
```

Managing Consents

A client application can enable its end users to view and manage the consents that they grant for data access by making calls to the following endpoints:

- `/metadata/v1/<userID>/consentHistory` – Retrieves consent history for the specified user ID.
- `/metadata/v1/<userID>/consents` – Retrieves, adds, or deletes a consent for a given application, action, purpose, and resource(s).
- `/metadata/v1/<userID>/consents/applications` – Retrieves a list of all applications to which the specified user ID has given consented.
- `/metadata/v1/<userID>/consents/resources` – Retrieves a list of all resources to which the specified user ID has given consented.

The Identity Broker installs the following scopes to access consent data:

`read_consents` – Enables reading the consents or consent history records for the specified user ID, and includes the following resources:

```
urn:unboundid:resources:broker_metadata:consents
urn:unboundid:resources:broker_metadata:consentHistory
```

`manage_consents` – Enables adding, updating, or deleting the consents for the specified user ID, and includes the following resources:

```
urn:unboundid:resources:broker_metadata:consents
```

Read Consent Examples

Request:

```
GET /metadata/v1/9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b/consents?application=MyApp
Host: <example.com>
Accept: application/json
Authorization: Bearer Aes-6SPszrDDpFxKuCdDqDxoZSdqAAAAAAAAAAB-sedGtKSBOaJdg3opJsRtLyqqF_k
uE92iiVFvi0LIqXYcjrqQK-6HVhqGUyWiDP84kpmZaMm9pestt4O2PVyVlWrd__6wa4rU_NLVelrleA
Content-Type: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: …
{
  "startIndex":0,
  "count":1,
  "totalResults":1,
  "data": [
    {
      "action": {
        "name":"Read",
         ... // other action properties
      },
      "actorCompositeKey": "9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b",
      "application": {
        "name":"MyApp",
         ... // other application properties
      },
      "ownerCompositeKey": "9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b",
      "purpose": {
        "name":"Any",
        "description":"Wild card that matches any purpose.",
        ... // other purpose properties
      },
      "resourceMap":{
        "2014-09-03T14:32:41.000+0000": [
          {
            "urn":"urn:example:resource:customer-profile",
            "name":"Customer Profile",
            ... // other resource properties
      },
            ... // other resources
        ],
        ... // other map entries
```

```
      }
    },
    ... // other consent entries
  ]
}
```

jQuery Example:

```
$.ajax({
type: "GET",
url: "https://<example.com>/metadata/v1/" + userId + "/consents?application=MyApp",
headers: { "Authorization": "Bearer " + accessToken },
dataType: "json",
success: function(data) {
// do something interesting with the returned consent records
  }
});
```

Read Consented Applications Examples

Request:

```
GET /metadata/v1/9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b/consents/applications
Host: <example.com>
Accept: application/json
Authorization: Bearer Aes-6SPszrDDpFxKuCdDqDxoZSdqAAAAAAAAAAB-sedGtKSBOaJdg3opJsRtLyqqF_k
uE92iiVFvi0LIqXYcjrqQK-6HVhqGUyWiDP8UUtLWN5YDssa4tV15fmSCpYZ7QNXycne0ODjJCUUJOQ
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
{
  "startIndex":0,
  "count":4,
  "totalResults":4,
  "data": [
    {
      "name":"MyApp",
      ... // other application properties
    },
      ... // other applications
  ]
}
```

jQuery Example:

```
$.ajax({
type: "GET",
url: "https://<example.com>/metadata/v1/" + userId + "/consents/applications",
headers: { "Authorization": "Bearer " + accessToken },
dataType: "json",
success: function(data) {
    // do something interesting with the returned applications
  }
});
```

Add Consent Examples

Request:

```
POST /metadata/v1/9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b/consents?application=MyApp&
purpose=Marketing&resource=urn%3Ascim%3Aschemas%3Acore%3A1.0%3Aemails.preferred
Host: <example.com>
Authorization: Bearer Aes-6SPszrDDpFxKuCdDqDxoZSdqAAAAAAAAAAB-sedGtKSBOaJdg3opJsRtLyqqF_k
uE92iiVFvi0LIqXYcjrqQK-6HVhqGUyWiDP8UUtLWN5YDssa4tV15fmSCpYZ7QNXycne0ODjJCUUJOQ
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...
{
  "action":{
    "name":"Read",
  ... // other action properties
},
"actorCompositeKey":null,
"application":{
  "name":"MyApp",
... // other application properties
},
"ownerCompositeKey":"9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b",
"purpose":{
  "name":"Marketing",
...// other purpose properties
},
"resourceMap":{
  "2014-09-04T16:14:10.985+0000":[
    {
      "urn":"urn:scim:schemas:core:1.0:emails.preferred",
      ... // other resource properties
    },
    ... // other resources
   ],
   ...// other map entries
  }
}
```

jQuery Example:

```
$.ajax({
  type: "POST",
  url: "https://<example.com>/metadata/v1/" + userId + "/consents?application=MyApp&purpo
se=Marketing&resource=urn%3Ascim%3Aschemas%3Acore%3A1.0%3Aemails.preferred",
  headers: { "Authorization": "Bearer " + accessToken }, contentType: "application/json"
  dataType: "json",
  success: function(consent) {
    // do something interesting with the returned consent record
  }
});
```

Revoke Consent Examples

Request:

```
DELETE /metadata/v1/9f8a23-78d5a9b2-2b46-40ed-9d0a-57963ef50d1b/consents?application=MyAp
p&purpose=Marketing&resource=urn%3Ascim%3Aschemas%3Acore%3A1.0%3Aemails.preferred
Host: example.com
Authorization: Bearer Aes-6SPszrDDpFxKuCdDqDxoZSdqAAAAAAAAAAB-sedGtKSBOaJdg3opJsRtLyqqF_k
uE92iiVFvi0LIqXYcjrqQK-6HVhqGUyWiDP8UUtLWN5YDssa4tV15fmSCpYZ7QNXycne0ODjJCUUJOQ
```

Response:

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

jQuery Example:

```
$.ajax({
  type: "DELETE",
  url: "https://<example.com>/metadata/v1/" + userId + "/consents?application=MyApp&purpo
se=Marketing&resource=urn%3Ascim%3Aschemas%3Acore%3A1.0%3Aemails.preferred",
  headers: { "Authorization": "Bearer " + accessToken },
  success: function(){
    // no data returned...
  }
});
```

Adding an Identity Provider Link to an Account

An application can provide the means to link a local Identity Broker account with an account at an external identity provider. There are two ways to do this, as outlined in the following sample flows. The choice of flow depends on the client application. In both cases, the end user should already be authenticated, and the application should possess a bearer token for the `manage_links` scope.

Note

The `redirect_uri` value 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>/metadata/v1/providers/<provider_name>/callback`.

For Server-Side Applications

This flow is designed for server-side web applications where the access token should not be exposed to the client.

The server-side application initiates the linking flow by sending a server-to-server GET request to the Identity Broker's Metadata API at the end user's `links/interactive` resource.

Request:

```
GET /metadata/v1/{userID}/links/interactive?provider=<idp_name>&flow=server&redirectUri=<
application_redirect_URI>
Authorization: Bearer <bearer token>
Accept: application/json
```

The Identity Broker responds with a URI containing a one-time IDP link code.

Response:

```
HTTP/1.1 302 FOUND
Location: https://<identity_broker>/metadata/v1/providers/link?code=<one-time_link_code>
```

The application should then redirect the web browser to the Identity Broker URI containing the link code from the previous response.

Request:

```
GET /metadata/v1/providers/link?code=<one-time_link_code>
```

If the code is valid, the Identity Broker responds by redirecting the web browser to the external identity provider. The `Location` value will vary depending upon the external identity provider type and its configuration with the Identity Broker.

Response:

```
302 FOUND
Location: https://<identity_provider>/oauth/authorize?response_type=code&client_id=<ident
ity_broker_client_id>&scope=openid+profile+email&state=XXX&redirect_uri=https://<identity_
broker>/metadata/v1/providers/<idp_name>/callback
```

At the external identity provider, the end user may be prompted to log in and to authorize the request. Once the OAuth 2.0 flow is complete at the external identity provider, the external identity provider will redirect the browser back to the IDP callback URI.

Request:

```
GET https://<identity_broker>/metadata/v1/providers/<idp_name>/callback
```

The Identity Broker will complete the linking process by saving identity provider linking data to the end user's profile, and then redirect the web browser to the application's redirect URI.

Response:

```
302 FOUND
Location: https://{application host}/<redirect_path>?statusCode=200&provider=<idp_name>&p
roviderUserId=<idp_userID>
```

Query parameters identifying the linking flow status, identity provider name, and the end user's unique ID at the identity provider are appended to the redirect URI as query parameters.

For Client-Side Applications

The second flow is designed for client-side or native applications, where the access token must be stored in a potentially untrusted client-side environment. This flow skips the initial REST call that initiates the linking process by generating a one-time code.

The client-side application initiates the flow by sending a GET request to the Identity Broker's Metadata API at the end user's `links/interactive` resource:.

Request:

```
GET /metadata/v1/{userID}/links/interactive?provider=<idp_name>&flow=client&redirectUri=<
application_redirect_URI>
Authorization: Bearer <bearer token>
Accept: application/json
```

The Identity Broker responds with a JSON document containing a single `redirectUrl` field. This response is provided rather than a 302 redirect response to avoid potential cross-origin request difficulties for JavaScript applications. The `redirectUrl` value depends upon the external identity provider type and its configuration with the Identity Broker.

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "redirectUrl": "https://<external identity provider>/oauth/authorize?response_type=code
&client_id=<identity_broker_client_id>&scope=openid+profile+email&state=XXX&redirect_uri=
https://<identity_broker>/metadata/v1/providers/<idp_name>/callback"
}
```

The client-side application then redirects the browser using a GET to the `redirectUrl` value. This redirects the end user to the external identity provider.

At the external identity provider, the end user may be prompted to log in and authorize the request. Once the OAuth 2.0 flow is complete at the external identity provider, the external identity provider will redirect the browser back to the provider's callback URI.

Request:

```
GET https://{identity_broker}/metadata/v1/providers/<idp_name>/callback
```

The Identity Broker completes the linking process by saving the identity provider linking data to the end user's profile, and then redirects the web browser to the application's redirect URI.

Response:

```
302 FOUND
Location: https://<application_host>/<redirect_path>?statusCode=200&provider=<idp_name>&p
roviderUserId=<idp_userID>
```

Query parameters identifying the linking flow status, identity provider name, and the end user's unique ID at the identity provider are appended to the redirect URI as query parameters.

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.

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 (see the UnboundID Identity Broker Administration Guide) 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.

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.

XACML Attribute	Attribute Value
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

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 pre-defined by the Identity Broker and will always begin with `urn:un-boundid:resources:broker_metadata:`. For a complete list of metadata resource URNs, see Accessing User Metadata.

Chapter 6: Reference Information

The functionality for authorization, authentication, and data access is well documented by the OpenID Connect, OAuth2, and SCIM foundations.

This chapter provides references to that documentation and documentation for using the Identity Broker API endpoints.

Documentation

Related Information

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)*

Reference Information

The following are useful references to information in this guide:

- **JavaScript Object Notation (JSON) and JSON Web Token (JWT)**. JSON is a serialized text-based data interchange format using name-value pairs and ordered or unordered lists of values as its data structure. JSON Web Token (JWT) is a string representing a set of claims (attributes) as a JSON object that is encoded in a JSON Web Signature (JWS), enabling the claims to be digitally signed.
- **OAuth2 Specification**. The OAuth 2.0 Authorization Framework (RFC 6749) is an open standard that enables client applications to obtain the authorization to access resources on behalf of the resource owner.
- **OAuth2 Bearer Token Specification**. The OAuth2 Authorization Framework: Bearer Token Usage specification (RFC 6750) describes how to use bearer tokens in HTTP requests to gain access to resources.
- **OpenID Connect Drafts**. The Identity Broker provides the libraries and software packages to fully function as a standalone OpenID Provider or resource server.
- **XACML 3.0 Specification**. The Policy Service is XACML 3.0-compliant and requires a working knowledge of its core concepts.
- **Cross-Origin Resource Sharing (CORS)**. Applications that make JavaScript requests to the Identity Broker should be registered with their trusted domains defined. The CORS specification is a W3C recommendation.
- **External Identity Provider Login**. The Identity Broker Server supports login through Google, Facebook, and OpenID Connect providers. Configuration information is included in the *UnboundID Identity Broker Administration Guide*.

Index