UnboundID® Identity Broker

Client Application Developer Guide

Version 4.1

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

Copyright

This document includes possible practices and procedures relating to securing and monitoring an UnboundID software infrastructure. It is intended only for security professionals. While reasonable efforts have been made to include best practices and helpful suggestions, it is the sole responsibility of the security professional to evaluate all such practices and procedures, to decide the best approach for the applicable unique environment, to implement such decisions related to securing software infrastructure, and for the results obtained. This document does not constitute "Documentation" as such term is defined in any agreement between UnboundID and any of its customers.

This distribution may include materials developed by third parties. Third-party URLs are also referenced in this document. UnboundID is not responsible for the availability of third-party web sites mentioned in this document. UnboundID does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. UnboundID will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources. “UnboundID” is a registered trademark of UnboundID Corporation. UNIX is a registered trademark in the United States and other countries, licenses exclusively through The Open Group. All other registered and unregistered trademarks in this document are the sole property of their respective owners.

Table of Contents

Preface

The UnboundID Identity Broker Client Application Developer Guide presents the general concepts and API information to interface with the UnboundID Identity Broker. We appreciate any feedback and requests for specific topics to cover in future revisions of this guide.

About This Guide

This guide presents both general concepts and specific decisions you can make to develop or modify client applications and resource servers to interface with the UnboundID Identity Broker.

You can read this guide...

- To become familiar with the UnboundID Identity Broker API.
- As an initial guide to deploy an UnboundID Identity Broker or UnboundID Privacy Suite.
- As a reminder of topics and configuration options to revisit periodically, even after your environment has been running smoothly in production for a while.

Audience

This guide is intended for software developers interested in developing applications using the UnboundID Identity Broker API. These developers must ensure that the

client application or resource servers properly access the OAuth2 and Policy Services of the UnboundID Identity Broker.

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

- ReSTful web services and principles
- JSON or XML serialization formats
- XACML 3.0
- OAuth2 specification (RFC 6749)
- OAuth2 Bearer Token specification (RFC6750)

Related Documentation

The UnboundID Identity Broker 4.1 comes with the following document set, available in the `docs` folder of the server.

- *UnboundID Identity Broker Installation Guide*
- *UnboundID Identity Broker Client Application Developer Guide*
- *UnboundID Identity Broker Reference Guide (HTML)*
- *UnboundID Identity Broker REST API Reference*
- *UnboundID Identity Broker Administration Guide (available Summer 2013)*
- *UnboundID Identity Broker Deployment Guide (available Summer 2013)*

UnboundID Identity Data Store 4.1:

- *UnboundID Identity Data Store Administration Guide*
- *UnboundID Identity Data Store Reference Guide (HTML)*

UnboundID Identity Data Proxy 4.1:

- *UnboundID Identity Proxy Administration Guide*
- *UnboundID Identity Proxy Reference Guide (HTML)*

UnboundID Identity Data Sync 4.1:

- *UnboundID Identity Data Sync Server Administration Guide*
- *UnboundID Identity Data Sync Server Reference Guide (HTML)*

UnboundID Metrics Engine 4.1:

- *UnboundID Metrics Engine Administration Guide*
- *UnboundID Metrics Engine Reference Guide (HTML)*

Chapter 1 Introduction

The UnboundID Identity Broker is a pure-Java server that features high performance, high scalability, and high security to power your Identity Platform. The server provides an OAuth2, Policy, and an Administration service, capable of handling millions of policy decisions per day. The Identity Broker provides multiple REST API endpoints to allow any client application and resource server to easily communicate with the Identity Broker.

This guide presents the topics required to help you configure your application and/or resource server to interface with the OAuth2, Policy, or Admin services.

About the UnboundID Identity Broker

The UnboundID Identity Broker is a high-performance, highly-scalable, highly secure centralized Policy engine and OAuth2 authentication/authorization service. The Policy engine provides XACML 3.0-based implementations of a Policy Administration Point (PAP) and Policy Decision Point (PDP) to authorize access by applications (e.g., client, mobile, social or partner apps) to resources that are subject to privacy policies.

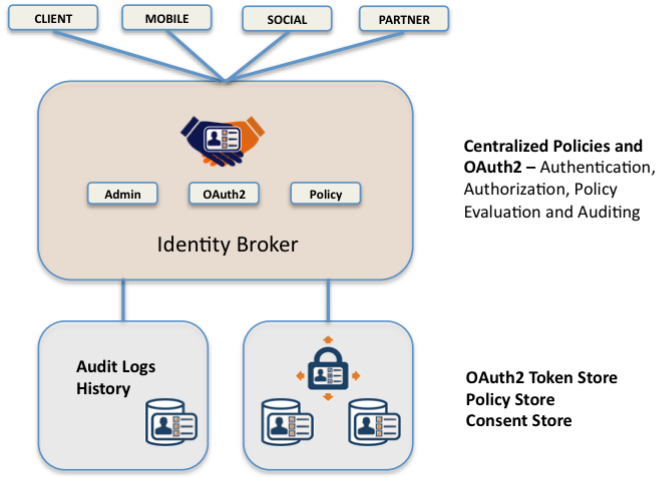


Figure 1-1. Identity Broker System

The Policy Engine authorizes any data access request against the company's data protection policies. A *policy* is a rule, or set of rules, that typically resolve an access control question, such as "Does application X have the ability to access a given user's profile?" A *policy set* is an ordered collection of policies that work together to derive a policy decision. Administrators have the ability to create new policies or policy sets using a Policy Template or create them from scratch. They can also disable existing policies, or test new policies against production data to ensure their validity before deploying them into production. All policies must be XACML 3.0-compliant and must be independent of other policies.

OAuth2 is a popular, open-standard authentication/authorization protocol that allows HTTP services, on behalf of a user or resource owner, to grant third-party access to their resources without sharing password and account information. The OAuth2 service lets the Identity Broker act as an authorization server, issuing an access token to the client application after user validation. The access token is then used to gain access to the user's account information stored at a resource server. Companies that have deployed their own OAuth2 server can use the Identity Broker as a Policy engine with the external OAuth2.

The Identity Broker also allows you to run data analytics to collect behavior trends about how your users' data is protected and accessed across the entire organization.

Identity Broker Features

The UnboundID Identity Broker is a full-featured server, providing OAuth2, Policy, and Admin services to power your privacy infrastructure. The Identity Broker has the following features:

- **Industry-Standard OAuth2**. The OAuth2 service provides a token-based authorization service, supporting all OAuth2 grant types outlined in RFC 6749: authorization code grant (for web server apps), implicit grant (for browser-based apps or mobile apps), and client credential grants (for application access). The Identity Broker's OAuth2 service can be disabled to allow for external OAuth2 implementations.
- **Policy Engine** The Policy service is a decision engine to determine whether client applications can be granted access to user resources and comply with company, national or industry policies. The policies are XACML 3.0-compatible and can be exported in XACML format for validation.
- **Governance Tags/Trust Level Evaluation**. The Identity Broker supports policy *governance tags*, which are attribute strings that are associated with both resources and applications. For example, a Policy Administrator can assign a HIPAA (Health Insurance Portability and Accountability Act) tag to a user's medical record resource to indicate that access to the medical record must be in compliance with HIPAA regulations. *Trust levels* are associated with both resources and applications. For example, a Policy Administrator can assign a configured trust level (e.g., "Medium" from a ranked set of trust levels of "Low", "Medium" or "High") to some resource. Any requesting application must have a trust level greater than or equal to the resource trust-level to be granted access to it. Policies can be constructed and filtered based on governance tags and/or trust levels.
- **Multiple REST API Endpoints**. The Identity Broker provides multiple REST API endpoints to interface with the OAuth2, Policy, and Admin engines via the HTTP protocol.
- **Testing Sandboxes**. The Identity Broker provides policy sandboxes to test policies against sample requests for pre-production testing. Also, administrators can enable a decision trace to log intermediate queries to troubleshoot policy rules.
- **Server SDK Extension Point**.The Identity Broker provides multiple extension points to allow extend its functionality. You can create your extensions using the UnboundID Server SDK. For example, if a request comes in to grant access to a resource for a client application, there may be a need for additional information about the user to verify the user's account or to fulfill a policy requirement before granting access. In this case, the Identity Broker can retrieve this additional attribute from an external database using an extension.
- **Data Analytics based on Audit and History Logs**. The Identity Broker provides the ability to store auditing and history logs so that the following questions can be answered: "When was consent given?", "When was it revoked?" and "Which systems have asked for access to what data?"

About the UnboundID Privacy Suite

The UnboundID Privacy Suite is a tightly integrated software solution enabling service providers to simultaneously access and protect customer data. Designed for tier-1 Telcos, the solution connects your mission critical data infrastructure to enhance and expand end-user experiences. The Privacy Suite consists of the following components:

- Telco-grade Identity Data Store
- High-performance Policy Engine
- OAuth2 Authorization Server
- Collection of stakeholder specific-applications that organize your data and make your policies actionable in real time
- Platform for integrating data consuming applications to the rules of the road

The Privacy Suite is deployed as three separate, run-time components: the web app layer, the middle-service layer, and the backend data layer. The web app layer consists of the administrative user interface and reference end-user interfaces that interact with the Identity Broker. At the heart of the service layer is the UnboundID Identity Broker, a robust and highly-scalable server that powers an OAuth2 authorization service, a Policy service, and an Administrative service for your Identity Infrastructure. The data layer consists of the UnboundID Identity Data Store server, which manages storage of end-user consent, OAuth2 tokens, configuration and other privacy domain data. The run-time components provide a robust and scalable, end-to-end privacy solution to power your Identity Infrastructure.

The Privacy Suite provides an easy way to deploy an Identity Infrastructure system out-of-the-box with the flexibility to integrate and customize the various components into your current deployment. The UnboundID Privacy Suite and UnboundID Identity Broker comes with REST APIs, such as the OAuth2 API used to customize token requests, validations, or revocations; the Admin API to customize policy creation and application registration; the Policy Definition Import API to configure how you want to export your policies into XACML; and other APIs for easy integration with your production environment.

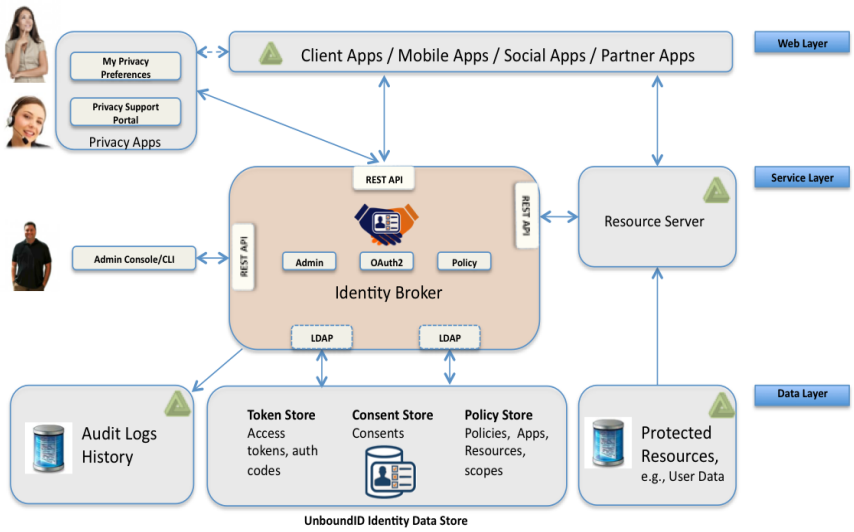


Figure 1-2. UnboundID Privacy Suite

The Privacy Suite comes with the Broker Console, a graphical administrative interface to manage your Privacy Suite. You can also build your own interface using the APIs provided with the Identity Broker. Command-line tools are also available for administrators who prefer scripting.

The OAuth2, Consent, and Policy data are stored in a backend set of UnboundID Identity Data Store servers. If you want to use your existing data stores, you can easily integrate this system with your current infrastructure. For example, if you want to store your OAuth2 tokens in another database, you can configure it via an extension.

About UnboundID

UnboundID Corp is a leading Identity Infrastructure Domain solutions provider with proven experience in large-scale identity data solutions with certified information privacy professional team members. The following are just a few reasons why UnboundID does it best:

- **Secure End-to-End Customer Data Privacy Solution**. UnboundID has developed a comprehensive integrated Identity Data 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 (at rest, in motion, in use).

- **Purpose-Built Identity Data Platform**. UnboundID has designed a solution to consolidate, secure, and expose customer consent-given identity data. The system provides unmatched security measures to protect sensitive identity data and maintain its visibility. The broad range of platform services include, policy management, cloud provisioning, federated authentication, data aggregation, and directory services.
- **Unmatched Performance across Scale and Breadth**. UnboundID has developed a solution that accommodates the three pillars of performance-at-scale: users, response time, and throughput. The system supports real-time data at large-scale consumer facing service providers today.
- **Support for External APIs**. UnboundID has developed solutions to interface with various external APIs to access a broad range of services to its products. These APIs include XACML 3.0, SCIM, LDAP, OAuth2 and very soon in a future release, OpenID Connect and SAML.
- **Leading Manageability, TCO, and ROI with Identity Management Domain Expertise**. UnboundID has developed a lightweight architecture that keeps hardware and people costs down with the ability to easily add new services. The Identity Data Platform has components that can exist as a complete solution or can be individually integrated within an existing production system with multiple RESTful API endpoints and plug-in points to interface with the server.

Chapter 2 Developing Apps

The UnboundID Identity Broker REST API allows you to integrate client applications using simple HTTP methods in JSON or XML formats. The API is ideal for developing web, mobile, social or external client/partner applications that request access to resources via the UnboundID Identity Broker. For example, the API allows applications to get a token and use it against a resource server or an application that uses the Policy Server, and passes it to the Identity Broker's Policy endpoint.

Before You Begin

The following prerequisites must be fulfilled so that you can successfully develop your applications.

- **Download cURL**. cURL is a command-line tool for transferring data using URL syntax, supporting the HTTP/HTTPS, LDAP/LDAPS protocols among others. cURL is a useful tool, but not required for app development.
- **Familiarize Yourself with Javascript Object Notation (JSON)**. JSON is a lightweight, serialized text-based data interchange format using name-value pairs and ordered or unordered lists of values as its data structure.
- **Read the OAuth2 Specification**. The OAuth 2.0 Authorization Framework (RFC6749) is a popular, open standard that allows client applications to obtain the authorization to access resources on behalf of the resource owner.
- **Read the OAuth2 Bearer Token Specification**. While you're at it, read the OAuth2 Authorization Framework: Bearer Token Usage specification (RFC 6750). The spec describes how to use bearer tokens in HTTP requests to gain access to resources.

- **Familiarize Yourself with XACML 3.0**. The Policy Service is XACML 3.0-compliant and requires a working knowledge of its core concepts.
- **Read the Spring OAuth Framework**. The Identity Broker uses Spring Security and the Spring OAuth Client library.

About the REST Endpoints

The Identity Broker provides multiple Representational State Transfer (REST) endpoints, fully configured and enabled out-of-the-box after you have installed the server. The Identity Broker Admin REST API supports JSON objects (`Content-Type=application/json`) as well as XML for endpoints requiring XACML formatted input.

The endpoints expose the RESTful services for accessing a set of resources through a fixed set of HTTP Create-Read-Update-Delete (CRUD) methods for all objects. The following HTTP request types are available for all Identity Broker Admin API resources:

- **POST**. Creates an object.
- **GET**. Retrieves an object.
- **PUT**. Updates an object.
- **DELETE**. Deletes an object.
- PATCH. Not supported in this release.
- HEAD, OPTIONS, TRACE. Not supported.

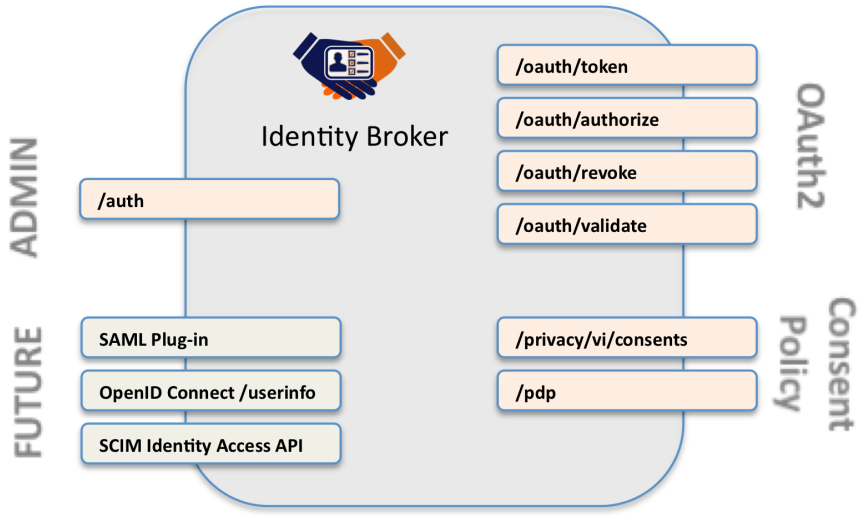


Figure 2-1. Identity Broker Endpoints

The endpoints are summarized as follows:

Endpoint	Description
/oauth/authorize	An endpoint to validate authorization requests, including prompting for end-user approval if required. The **`/oauth/authorize`** endpoint supports all of the parameters outlined in RFC 6749 (http://tools.ietf.org/html/rfc6749).
/oauth/token	An endpoint for client applications to get OAuth access tokens. The **`/oauth/token`** endpoint supports all of the parameters outlined in RFC 6749 (http://tools.ietf.org/html/rfc6749) and RFC6750 (http://tools.ietf.org/html/rfc6750) for bearer tokens.
/oauth/validate	An endpoint that validates an OAuth access token in response to a request for resources. This endpoint is not defined by the OAuth2 spec, but is UnboundID proprietary. The endpoint allows a resource server to validate an access token, letting you pass in an access token and getting back the userName, list of authorities, token scopes, client ID, and issued time in seconds.
/oauth/revoke	An endpoint that provides a means for an end-user or authorized agent to revoke a previously granted access token. It lets you remove that token at any time. The endpoint is based on http://www.ietf.org/id/draft-ietf-oauth-revocation-07.txt.
/pdp	An endpoint that receives or returns XACML requests and responses between the resource server and the Identity Broker.
/privacy/v1/consents	An endpoint that receives or returns consent requests and responses between the user agent and the Identity Broker.
/auth	An endpoint that REST APIs for administering the Identity Broker. The Broker Console GUI and the command-line tools (**`broker-admin`**, **`consent-admin`**, **`evaluate-policy`**, **`oauth2-request`**) are built as clients of the Admin API. The Admin endpoint has direct access via the REST API to the runtime objects used by the Identity Broker. The Policy Engine uses the endpoint to authorize administrative actions and import XACML policies into the Identity Broker. The Admin API can also query the Access History store to retrieve a history of all requests handled by the Policy Engine along with their decisions through the **`/auth`** endpoint.

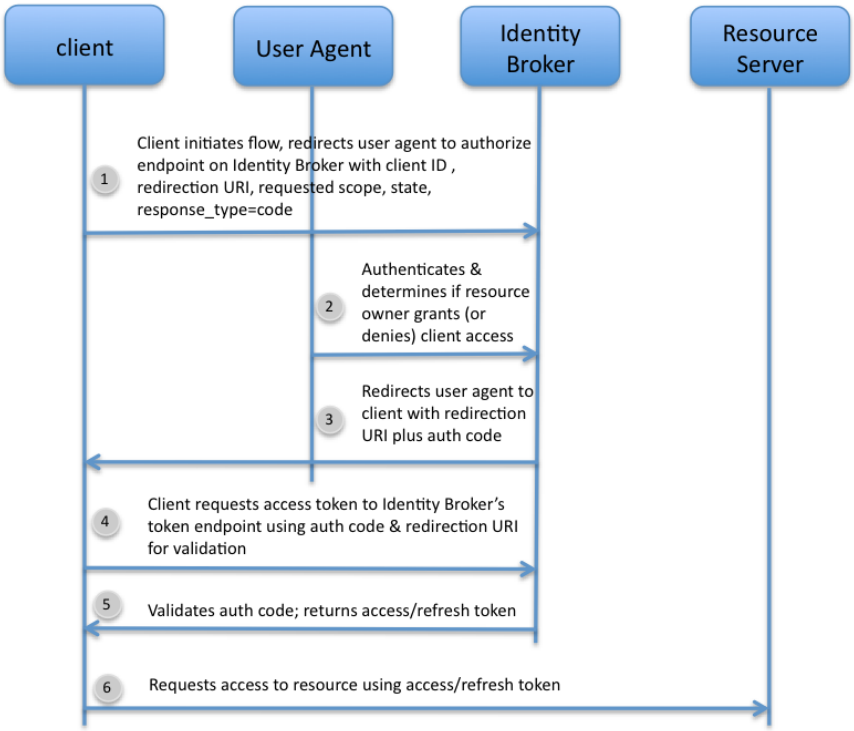
Refer to the *UnboundID Identity Broker REST API Reference* for more information at **`<server-root>/docs/restapi/index.html`**.

About the Supported OAuth2 Flows

The Identity Broker's OAuth services supports all of the OAuth2 flows, described as follows:

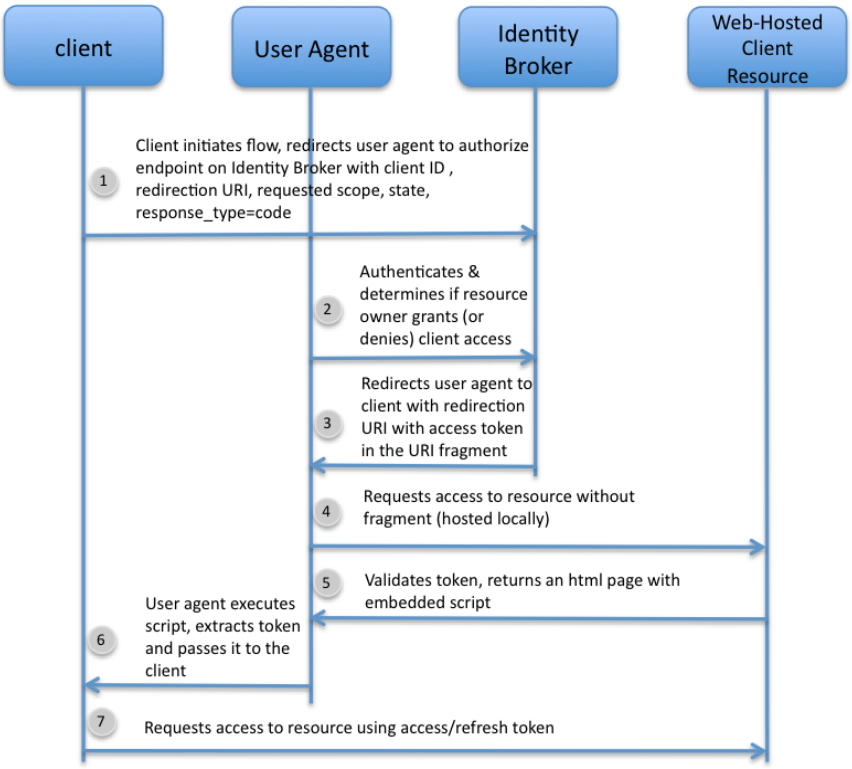
- **Authorization Code Grant**. The Authorization Code Grant flow is designed for web applications and is a redirection-based flow (RFC6949). The flow begins with the client redirecting the user agent to the authorization endpoint of

the Identity Broker. The user agent authenticates with the Identity Broker to determine if the resource owner grants or denies the client access to the resource. If access is granted, the Identity Broker returns a redirection URI with an authorization code and then redirects the user agent back to the client. The client authenticates to the Identity Broker and then requests an access token using the authorization code and the redirection URI. The Identity Broker validates the authorization code and returns an access or refresh token. The client then makes a request to access the resource on the resource server using the access/refresh token.

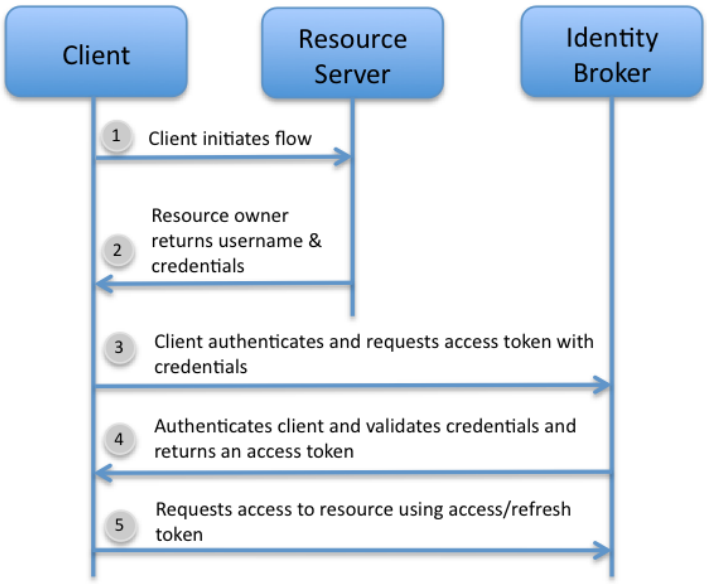


- **Implicit Grant Flow**. The Implicit Grant Flow is designed for browser or mobile applications. The flow begins in much the same way as the authorization grant flow, but differs when the Identity Broker returns a URI fragment with an embedded access token. The user agent makes a request to a web-hosted client resource. The web-hosted client resource returns an HTML page with an embedded script. The user agent executes the script to extract the access token, which is

then passed to the client. The client secret is not used with the request.

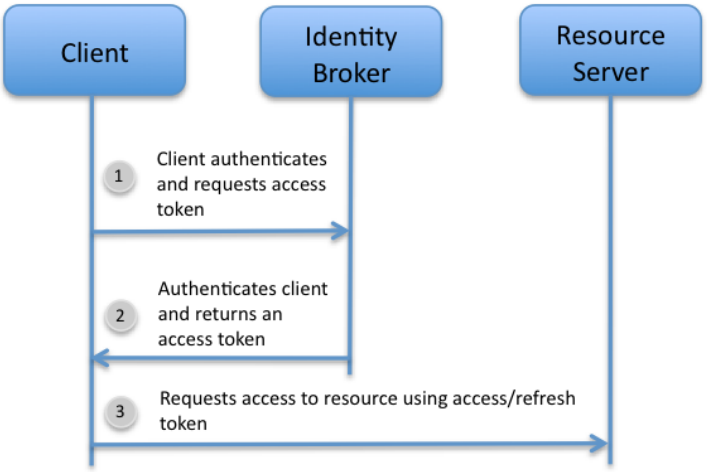


- **Resource Owner Password Flow**. The Resource Owner Password Flow allows a user to login with her username/password access for an access token. This flow should be used for trusted clients, such as a native application that needs to log into your company's desktop or mobile app.



- **Client Credentials Flow**. The Client Credentials Flow allows a client's server to exchange client ID and secret for an access token. This flow allows 3rd-party

applications direct access to resources.



OAuth2 Service APIs

The Identity Broker's OAuth2 service is based on the Spring Security OAuth2 implementation. We recommend that you cross reference this OAuth2 information with RFC6749 at http://tools.ietf.org/html/rfc6749.

The following section provides a general introduction to the Identity Broker's OAuth2 Service APIs.

The Identity Broker provides the following endpoints:

- **/oauth/authorize**. OAuth2 standard endpoint for authorization.
- **/oauth/token**. OAuth2 standard endpoint for token requests.
- **/oauth/validate**. An endpoint used to validate a token and obtain information about it.
- **/oauth/revoke**. An endpoint used to revoke a token. The endpoint is based on the draft, **`http://www.ietf.org/id/draft-ietf-oauth-revocation-07.txt`**.

/oauth/authorize

The **`/oauth/authorize`** endpoint accepts authorization requests and handles user approval if the grant type is **`auth_code`**. For example, the endpoint is used for the following operations:

- **Authorizing or denying consents** during the OAuth2 authorization code and implicit grant flows.
- **Requesting an authorization code** using the OAuth2 authorization code grant type.
- **Requesting an access token** using the OAuth2 implicit grant type.

The endpoint supports the following special parameters, **`prompt`** and **`access_type`**.

- **prompt**. Specifies whether the Identity Broker should prompt the end user for re-authentication and consent. Values are space delimited, case sensitive list of ASCII strings. The supported values are: "none" and "consent".
  - **None**. Indicates that the Identity Broker must NOT display any authentication or consent user interface pages. An error is returned if the end user is not already authenticated or the client does not have pre-configured consent for the requested claims. This can be used as a method to check for existing authentication and/or consent.
  - **Consent**. Indicates that the Identity Broker MUST prompt the end-user for consent via a consent user interface page before returning information to the client.
- **access_type=offline**. Used when requesting an authorization code using GET. This allows the authorization approval page to display text to the resource owner indicating that the application is requesting offline access to resources. To receive an access token, the client must again provide this parameter to the token endpoint when requesting an access token. The **`access_type`** parameter may not be used with the implicit grant type and is only used when **`response_type=code`**.

The following example HTTP Header request forces the consent approval UI to show regardless if the user has already consented to the request.

```
GET oauth/authorize?response_type=code&client_id=0d5e5af7-420c-4241-8cff-0cfd9d806e59&prompt=consent HTTP/1.1
```

The following example HTTP Header request returns an authorization code that may be exchanged for an access token *and* a refresh token:

```
GET oauth/authorize?response_type=code&client_id=0d5e5af7-420c-4241-8cff-0cfd9d806e59&access_type=offline HTTP/1.1
```

The **`/oauth/authorize`** endpoint also supports the following form parameters, which are used when granting or denying a consent from the authorization approval page. They must be used with POST:

- **user_oauth_approved_scopes**. Specifies a space-delimited list of scope names that are explicitly approved when **`user_oauth_approval=true`**. If not provided

or no value is present, then all scopes are approved.

- **user_oauth_approval**. Specifies whether the end user has consented to the authorization request.

For example, the following request approves only scopes "read-scope" and "write-scope" in the request:

```
POST /oauth/authorize?user_oauth_approval=true&user_oauth_approved_scopes=read-
scope%20write-scope HTTP/1.1
```

The following approves all scopes in the request:

```
POST /oauth/authorize?user_oauth_approval=true HTTP/1.1
```

The default authorization approval template is located at **<server-root>/config/pages/templates/oauth-approval.vm**. An administrator may change this location using the **authorization-approval-template** property of the OAuth Service configuration object using the **dsconfig** tool.

Spring Security handles authentication to this endpoint using either form or Basic authentication. There will be no other access control enforced by Spring other than that the user must be authenticated.

/oauth/token

The **/oauth/token** endpoint accepts access token and refresh token requests.

The following example is taken from RFC 6749, section 4.4.2 and makes a token request to the endpoint over TLS:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Accept: application/json
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

If the token request is authorized, the 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":"example-scope1 example-scope2"
}
```

The client must be authenticated with Spring Security using Basic authentication. There will be no other access control enforced by Spring Security other than requiring that the client application be authenticated.

/oauth/validate

The **`/oauth/validate`** endpoint is used by resource servers to validate that an access token grants access to resources requested by a client application.When a client application wishes to access a resource from a resource server on behalf of a resource owner, it presents an OAuth access token (also called a "bearer token") in the HTTP Authorization header, as described in RFC6750. To validate that the access token should in fact grant access to the requested resource, the resource server submits the token to the Identity Broker's **`/oauth/validate`** endpoint, which returns a response including information about the resource owner, scopes and token validity.

The Identity Broker only supports the POST HTTP request type of the form parameter. The resource server should perform a POST to the validation endpoint with the following parameters:

- **token**. The access token provided by the client.

The following example validates a token:

```
POST /oauth/validate HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Token=2YotnFZFEjr1zCsicMWpAA
```

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

- **username**. Specifies the username of the resource owner.
- **authorities**. Specifies the authorities or roles granted to the client. See "About the Identity Broker Roles and Scopes" on page 27.
- **scope**. Specifies the scopes to which the client is granted.
- **issued_at** Specifies the timestamp when the token is issued.
- **expires_in**. Specifies the date-time when the token expires.

The Identity Broker returns the following response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "username": "admin",
  "authorities": [
  "ROLE_READ_OWN_CONSENT",
  "ROLE_READ_OWN_ACCESSHISTORY",
  "ROLE_READ_OWN_CONSENTHISTORY",
```

```
  "ROLE_USER",
  "ROLE_CREATE_OWN_CONSENT",
  "ROLE_BROKER_ADMIN",
  "ROLE_DELETE_OWN_CONSENT"
  ],
  "scope": [
  "test"
  ],
  "client_id": "0d5e5af7-420c-4241-8cff-0cfd9d806e59",
  "issued_at": "20130418201219Z",
  "expires_in": 0
}
```

The resource server acts as a client and needs to be registered with the Identity Broker as an application. The endpoint uses stateless Basic authentication using pre-shared client credentials to access it. There will be no other access control enforced by Spring Security other than requiring that the client application be authenticated.

/oauth/revoke

The **`/oauth/revoke`** endpoint is used to allow clients to send an HTTP POST request to the Identity Broker to revoke access or refresh tokens, for example, when the client logs out or uninstalls the application. The endpoint is based on http://www.ietf.org/id/draft-ietf-oauth-revocation-07.txt.

During the revocation process, the Identity Broker validates the client credentials, and then verifies that the client making the revocation request 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; thus, allowing the Identity Broker to clean up session data.

The Identity Broker only supports the POST HTTP request type of the form parameter. It does not support the **`token_type_hint`** or **`callback`** request types as stipulated in the draft.

For example, the client should perform a POST to the **`/oauth/revoke`** endpoint with the following parameters:

- **token**. The access token provided by the client.

The following example revokes a token:

```
POST /oauth/revoke HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Token=2YotnFZFEjr1zCsicMWpAA
```

If the operation is successful, the Identity Broker responds with the HTTP status code 200. After this revocation, the client must not use the token again. If the client

attempts to resend a revoke request with a now invalid token, the servers returns HTTP status code 400, which differs from the draft.

The following response is returned:

```
HTTP/1.1 200 OK
```

The client must already be authenticated with Spring Security using Basic authentication. There will be no other access control enforced by Spring Security.

Application Registration

An application can be registered for the following roles:

- **policy**. Refers to the XACML Subject of the Policy Service functions
- **oauth2**. The OAuth2 client. This requires that the **`client_id`**, **`client_secret`**, and **`redirect_uri`** be defined.

The following example shows an application registration that fulfills both roles.

Register an application:

```
POST /auth/api/v1/applications
Content-type: application/json

{
  "name" : "example-app",
  "description" : "This is an example application",
  "role" : [ "oauth2", "policy" ],
  "enabled" : true,
  "redirect_uris" : [ "https://example.com/redirect" ],
  "trustLevel" : medium,
  "applicationGroups" : [],
  "tags" : []
}
```

Note that the **`client_id`** or **`client_secret`** is not present in the request but is assigned by the Identity Broker in the response.

Response:

```
Content-type: application/json

{
  "id" : "1",
  "name" : "example-app",
  "description" : "This is an example application",
  "role" : [ "oauth2", "policy" ],
  "enabled" : true,
```

```
  "redirect_uris" : [ "https://example.com/redirect" ],
  "trustLevel" : medium,
  "applicationGroups" : [],
  "tags" : [],
  "client_id" : "generated_id",
  "client_secret" : "generated_password"
}
```

In the response, the Identity Broker assigns a unique ID to the application object. The ID is used in subsequent API requests. For example, to update the application definition, you would use:

Update an application:

```
PUT  /auth/api/v1/applications/8AP
```

Or, to delete the application, you would use:

Delete an application:

```
DELETE /auth/api/v1/applications/8AP
```

Policy Service APIs

The Policy Service APIs are as follows:

Policy Authorization (Policy Decision):

```
POST /pdp/v1/authorization
Content-type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
...
</Request>
```

Response:

```
Content-type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
...
</Response>
```

Request Detailed Trace of the Policy Evaluation. An UnboundID-defined field DecisionTrace is added to the standard XACML Result element.

```
POST /pdp/v1/authorization?traceEnabled=true
```

```
...
```

Response:

```
Content-type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  <Result>
  ...
    <DecisionTrace>
    ...
    </DecisionTrace>
  </Result>
</Response>
```

User Consent APIs

The user consent management APIs are as follows:

Retrieve Owner:

```
GET /privacy/v1/consents?self
```

Retrieve Applications for Owner:

```
GET /privacy/v1/consents/self/applications
```

Retrieve Consents for Owner and Application:

```
GET /privacy/v1/consents/self/consented/applications
```

Delete Consents for Owner and Application:

```
DELETE /privacy/v1/consents/self/consented/applications
```

Delete Consents for Owner, Application and Purpose:

```
DELETE /privacy/v1/consents/self/consented/applications/purpose/1
```

Delete Consents for Owner, Application, Purpose and Resource:

```
DELETE /privacy/v1/consents/self/consented/applications/purpose/1/resource/1
```

Delete Consents for Owner, Application and Resource:

```
DELETE /privacy/v1/consents/self/consented/applications/resource/1
```

Additional Query Parameters:

- **sortOrder**
  - ascending
  - descending

Administration Service APIs

The UnboundID Identity Broker comes with the Broker Console GUI that accesses the administration objects in the Policy data store using the Identity Broker's REST endpoints. The following endpoints are available for those who want to create their own Admin interface or integrate the functionality into their existing systems:

- /auth/api/v1/applications
- /auth/api/v1/actions
- /auth/api/v1/tags
- /auth/api/v1/trustLevels
- /auth/api/v1/policies
- /auth/api/v1/policySets
- /auth/api/v1/purposes
- /auth/api/v1/resources
- /auth/api/v1/resourceAliases
- /auth/api/v1/scopes

The following operations are supported by the admin resource endpoints:

- **POST**. Creates a new admin resource.
- **GET**. List existing resources or retrieves a specific resource.
- **PUT**. Replace an existing resource.
- **DELETE**. Delete an existing resource.

Note that the UnboundID Identity Data Store automatically creates a short unique numeric identifier and assigns it to any newly-created object. The identifier is used in subsequent API request URIs to identify the object.

The API also identifies policies by the **`PolicyId`** and policy sets by the **`PolicySetId`** defined in the XACML Policy. The XML for the policy or policy set is embedded in the JSON requests and responses.

Import New Policy:

```
POST /auth/api/v1/policies
Content-type: application/json
```

```
{
  "name" : "Example Policy",
  "enabled" : true,
  "policyType" : "
    <?xml version=\"1.0\" encoding=\"UTF-8\"?>
       <Policy xmlns=\"urn:oasis:names:tc:xacml:3.0:core:schema:wd-17\"
      PolicyId=\"urn:unboundid:policy:Example\"
         Version=\"1\"
       ...
       </Policy>"
}
```

Response:

```
Content-Type: application/json

{
  "policyId" : "urn:unboundid:policy:Example",
  "name" : "Example Policy",
  "enabled" : true,
  "policyType" : "
  <?xml version=\"1.0\" encoding=\"UTF-8\"?>
  <Policy xmlns=\"urn:oasis:names:tc:xacml:3.0:core:schema:wd-17\"
       PolicyId=\"urn:unboundid:policy:Example\"
    Version=\"1\"
    ...
  </Policy>"
}
```

Delete Policy:

```
DELETE /auth/api/v1/policies/urn:unboundid:policy:Example
```

List Policies:

```
GET /auth/api/v1/policies
```

Response:

```
Content-Type: application/json

{
 "policies" :
   [
     {
       "policyId" : "urn:unboundid:policy:Example",
       "name" : "example_policy",
       "enabled" : true
     },
     ...
   ]
}
```

Retrieve Policy:

```
GET /auth/api/v1/policies/urn:unboundid:policy:Example
```

Response:

```
Content-Type: application/json

{
 “policyId” : “urn:unboundid:policy:Example”,
 “name” : “Example Policy”,
 “enabled” : true,
 “policyType” : “
   <?xml version=\"1.0\" encoding=\"UTF-8\"?>
   <Policy xmlns=\"urn:oasis:names:tc:xacml:3.0:core:schema:wd-17\"
       PolicyId=\"urn:unboundid:policy:Example\"
       Version=\"1\"
  ...
   </Policy>”
}
```

Replace Existing Policy. Enable/disable existing policy:

```
PUT /auth/api/v1/policies/urn:unboundid:policy:Example
Content-type: application/json

{
 “name” : “Updated Example Policy”,
 “enabled” : false,
 “policyType” : “
   <?xml version=\"1.0\" encoding=\"UTF-8\"?>
   <Policy xmlns=\"urn:oasis:names:tc:xacml:3.0:core:schema:wd-17\"
       PolicyId=\"urn:unboundid:policy:Example\"
       Version=\"1\"
   ...
   </Policy>”
}
```

Response:

```
Content-Type: application/json

{
  “id” : “urn:unboundid:policy:Example”,
  “name” : “Updated Example Policy”,
  “enabled” : false,
  “policyType” : “
       <?xml version=\"1.0\" encoding=\"UTF-8\"?>
    <Policy xmlns=\"urn:oasis:names:tc:xacml:3.0:core:schema:wd-17\"
      PolicyId=\"urn:unboundid:policy:Example\"
      Version=\"1\"
         ...
       </Policy>”
}
```

Authentication using Spring Security

Both end users and client applications must authenticate to the Identity Broker with Spring Security, which has a very flexible **`AuthenticationProvider`** interface that handles the session management. Authentication sources are mapped to the

authorization endpoint (i.e., **`/oauth/authorize`** for LDAP user-store based authentication or **`/oauth/sso/authorize`** for enterprise SSO based authentication).

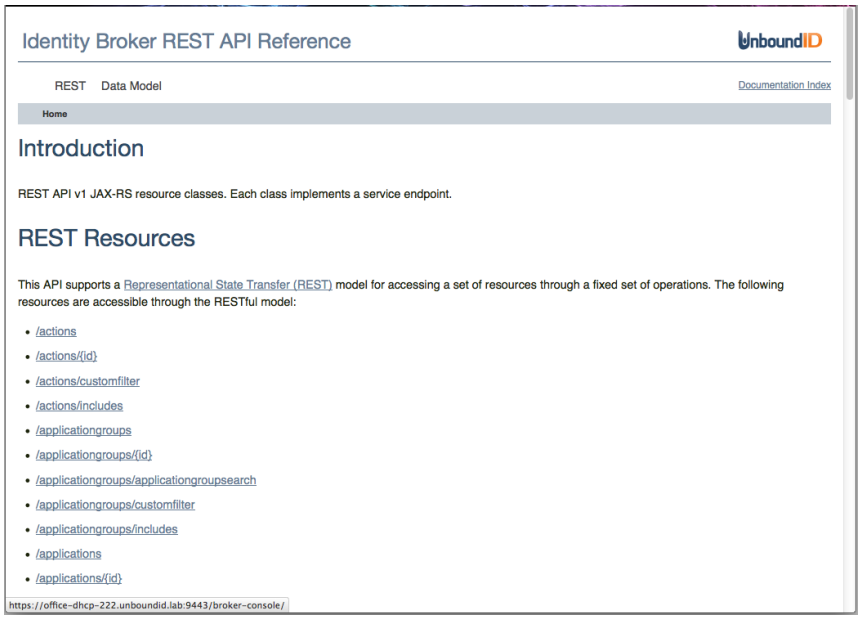
The Identity Broker's LDAP user-store based authentication source may be implemented using Spring Security's built-in LDAPAuthenticationProvider, which uses JNDI. However, UnboundID has implement its own components (i.e., IdentityMapper, LDAP SDK for Java) that is available to our customers. For form based authentication, you will also need to implement a JSP login page that is customizable.

Enterprise SSO variants can be integrated using a pluggable interface using Spring Security's AuthenticationProvider and UserDetailsService APIs. These interfaces and its dependencies should be configured in the extension XML to allow maximum flexibility and then referenced by the UnboundID configuration framework.

Developing the Client Application

The process is fairly straightforward as follows:

1. First, you must install and configure your UnboundID Identity Broker server. See the UnboundID Identity Broker Installation Guide.
2. View the Identity Broker REST API Reference. Open the file, located in the Identity Broker's installation: **`docs/restapi/index.html`**.



3. As a client application, you must pre-register the app on the Identity Broker. This is out-of-scope of this guide, but your service provider can assist you with this process.
4. The client app needs to know what scopes are available to it. The service provider should publically expose these scopes, so that you can hard-code these into your application. The scopes are not provided in the Identity Broker API.
5. Point to the Spring OAuth documentation to use the OAuth Client Library. The OAuth Client library works seamlessly with the Identity Broker (i.e., an OAuth server), providing high-level abstractions to manage the OAuth flows. You can also use any REST client of your own to construct the OAuth requests.
6. Secure your bearer tokens. Don't expose them to the user. They should stay in the application process. Applications will need to respect the token expiration and strive to refresh the token before its expiration date.

Chapter 3 Reference

The following chapter provides general reference about various files and components of the UnboundID Identity Broker.

Unsupported XACML Features

The UnboundID Identity Broker features a policy service based on the XACML 3.0 specification (http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf). The Identity Broker supports only those XACML 3.0 features that are necessary for optimized policy decisions for your Identity infrastructure. The following XACML 3.0 features are *not* supported:

- **No support for embedded XML content in a request**. In typical XACML deployments, embedded XML content is used to provide additional attributes to an incoming XACML request, needed to make an access decision. For example, these additional attributes could be captured consent information needed to complete a decision. In terms of security, the PEP would have to add the embedded XML content when formulating the request, rather than just arriving from the requesting application. UnboundID's solution is to have the Identity Broker retrieve these additional attributes directly from the persistent store and not convert it into XML to be fully XACML-compliant. As a result, the following XACML elements related to XML processing have *not* been implemented:
  - <PolicyDefaults> and <PolicySetDefaults>
  - <XPathVersion>
  - <AttributeSelector>
  - <Content>
  - XPath functions **`xpath-node-count`**, **`xpath-node-equal`**, and **`xpath-node-match`**.

- **No support for Obligations or Advice**. Obligations and Advice are additional actions, specified by policy, which either must (in the case of Obligations) or may (in the case of Advice) be taken by the PEP whenever a particular policy decision is rendered. An example might be to send an email to an individual every time permission is granted to access some resource owned by that person. The following XACML elements related to Obligations and Advice are not implemented:
  - <Obligations>
  - <Advice>
  - <AttributeAssignment>
  - <ObligationExpressions> and <ObligationExpression>
  - <AdviceExpressions> and <AdviceExpression>
  - <AttributeAssignmentExpression>
- **No support for versioning of Policies**. XACML incorporates the idea of maintaining multiple versions of a policy, e.g. policy "X" version 1.0 and policy "X" version 2.0. A policy set then can specify, via a reference, which version of policy 'X' is to be applied when evaluating a request. Currently, our policy store only allows for a single instance of policy X to be stored, and it does not support referencing a particular version of that policy. The following XACML elements related to versioning are not supported:
  - <VersionMatchType> in PolicyIDReference or PolicySetIDReference elements
- **Limited Support for Multi Requests**. XACML specifies several ways (i.e., nodes identified by scope, nodes identified by XPath, repeated <Attributes> categories) that a request for multiple decisions can be contained within a single request context, described in the XACML Multiple Decision Profile document which can be found at http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cd-03-en.pdf. Our implementation only supports one version of a multiple-decision request by using multiple <Attributes> of the same category in the request. In addition, we further restrict this feature by only supporting multiple instances of the Resources category. As a result the following XACML elements are not supported:
  - <MultiRequests>
  - <RequestReference>
  - <AttributesReference>
  - CombinedDecision attribute of the <Request> element
  - xml:id attribute of the <Attributes> element
- **No support for Attribute Issuer or Policy Issuer**. These features are used in Policy administration and allow for the writing of Policies that determine which other Policies should be used when evaluating a request. For example, a request may be subject only to policies whose issuer (author) are from some trusted

source. This is a second-order feature and not relevant for environments where all Policies are equal as to their trustworthiness. The following XACML elements related to issuers are not supported:

  - <PolicyIssuer>
  - Issuer attribute of the <Attribute> element

- **No support for Policy and Rule Combiner Parameters**. A policy-combining algorithm is a rule for how the decisions rendered by multiple applicable policies are to be combined in order to form an ultimate decision by a Policy Set or the PDP as a whole. Similarly, a rule-combining algorithm is a rule for how the decisions rendered by multiple rules within a single policy are to be combined. There are several standard such algorithms specified by XACML, all of which are supported by the UnboundID Identity Broker. The Policy and Rule Combiner Parameters are relevant only if custom rule-combining or policy-combining algorithms are in effect. Since the Identity Broker does not currently support adding custom rule-combining or policy-combining algorithms, XACML elements for the associated Combiner Parameters are not supported:

  - <CombinerParameters>
  - <RuleCombinerParameters>
  - <PolicyCombinerParameters>
  - <PolicySetCombinerParameters>

About the Identity Broker Roles and Scopes

Authentication is accomplished on the Identity Broker through Spring Security, which is enabled on the HTTPS Connection Handler. Only one Spring Security HTTP Servlet Extension may be used with both the HTTP and HTTPS connection handlers per server instance, so we recommend using it with the HTTPS connection handler.

The Identity Broker defines Admin Roles and scopes to perform operations using the Identity Broker REST APIs. The Identity Broker supports a single "Default" scope that all admin apps and clients use. It indicates that the user has successfully authenticated. The operations allowed by the authorities/roles that the user has determines the operations that the admin can carry out.

The roles must be populated by the Spring AuthenticationProvider at login time and are stored with the access token. The Identity Broker pulls the granted authorities from a configurable attribute on the user entry and group entries of which the user is a member. The default attribute is "id-broker-admin-privilege".

```
dn: uid=user.341,ou=Users,ou=Identity Broker,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
givenName: Anette
sn: Associates
cn: Anette Associates
initials: AKA
employeeNumber: 341
uid: user.341
mail: user.341@example.com
userPassword: password
telephoneNumber: +1 999 002 6412
homePhone: +1 003 008 0090
pager: +1 609 134 0692
mobile: +1 470 169 0659
street: 52513 Washington Street
l: Harrisonburg
st: MD
postalCode: 98130
postalAddress: Anette Associates$52513 Washington Street$Harrisonburg, MD
98130
description: This is the description for Anette Associates.
id-broker-admin-privilege: create_action
id-broker-admin-privilege: read_action
id-broker-admin-privilege: update_action
id-broker-admin-privilege: read_scope
id-broker-admin-privilege: read_application
id-broker-admin-privilege: read_tag
id-broker-admin-privilege: update_tag
```

The Spring Secuirty automatically adds the "ROLE_" prefix and uses this internally to represent a "SimpleGranted Authority". The values of the **`id-broker-admin-privilege`** attribute can be "create_action" or "admin" or "read_purpose". There are CRUD roles for every resource type, plus some special roles at the end for the history, policy decision point, and authorization log history endpoints.

By default all users will have these five authorities (implicitly):

- ROLE_CREATE_OWN_CONSENT
- ROLE_READ_OWN_CONSENT
- ROLE_DELETE_OWN_CONSENT
- ROLE_READ_OWN_CONSENTHISTORY
- ROLE_READ_OWN_ACCESSHISTORY

Authorities

- ROLE_CREATE_ACTION
- ROLE_READ_ACTION
- ROLE_UPDATE_ACTION
- ROLE_DELETE_ACTION

- ROLE_CREATE_APPLICATION
- ROLE_READ_APPLICATION
- ROLE_UPDATE_APPLICATION
- ROLE_DELETE_APPLICATION

- ROLE_CREATE_APPLICATIONGROUP
- ROLE_UPDATE_APPLICATIONGROUP
- ROLE_DELETE_APPLICATIONGROUP

- ROLE_CREATE_POLICY
- ROLE_READ_POLICY
- ROLE_UPDATE_POLICY
- ROLE_DELETE_POLICY

- ROLE_CREATE_POLICYSANDBOX
- ROLE_READ_POLICYSANDBOX
- ROLE_UPDATE_POLICYSANDBOX
- ROLE_DELETE_POLICYSANDBOX

- ROLE_CREATE_POLICYSET
- ROLE_READ_POLICYSET
- ROLE_UPDATE_POLICYSET
- ROLE_DELETE_POLICYSET

- ROLE_CREATE_POLICYTEMPLATE
- ROLE_READ_POLICYTEMPLATE
- ROLE_UPDATE_POLICYTEMPLATE
- ROLE_DELETE_POLICYTEMPLATE

- ROLE_CREATE_PURPOSE
- ROLE_READ_PURPOSE
- ROLE_UPDATE_PURPOSE
- ROLE_DELETE_PURPOSE

- ROLE_CREATE_REQUEST
- ROLE_READ_REQUEST
- ROLE_UPDATE_REQUEST
- ROLE_DELETE_REQUEST

- ROLE_CREATE_RESOURCE
- ROLE_READ_RESOURCE
- ROLE_UPDATE_RESOURCE
- ROLE_DELETE_RESOURCE

- ROLE_CREATE_RESOURCEALIAS
- ROLE_READ_RESOURCEALIAS
- ROLE_UPDATE_RESOURCEALIAS
- ROLE_DELETE_RESOURCEALIAS

- ROLE_CREATE_RESOURCEGROUP
- ROLE_READ_RESOURCEGROUP
- ROLE_UPDATE_RESOURCEGROUP
- ROLE_DELETE_RESOURCEGROUP

- ROLE_CREATE_SCOPE
- ROLE_READ_SCOPE
- ROLE_UPDATE_SCOPE
- ROLE_DELETE_SCOPE

- ROLE_CREATE_TAG
- ROLE_READ_TAG
- ROLE_UPDATE_TAG
- ROLE_DELETE_TAG

- ROLE_CREATE_TRACEFILTER
- ROLE_READ_TRACEFILTER
- ROLE_UPDATE_TRACEFILTER
- ROLE_DELETE_TRACEFILTER

- ROLE_CREATE_TRUSTLEVEL
- ROLE_READ_TRUSTLEVEL
- ROLE_UPDATE_TRUSTLEVEL
- ROLE_DELETE_TRUSTLEVEL

- ROLE_CREATE_OWN_CONSENT
- ROLE_CREATE_OTHER_CONSENT
- ROLE_READ_OWN_CONSENT
- ROLE_READ_OTHER_CONSENT
- ROLE_DELETE_OWN_CONSENT
- ROLE_DELETE_OTHER_CONSENT

- ROLE_READ_OWN_CONSENTHISTORY
- ROLE_READ_OTHER_CONSENTHISTORY

- ROLE_READ_OWN_ACCESSHISTORY
- ROLE_READ_OTHER_ACCESSHISTORY

- ROLE_INVOKE_PDP

- ROLE_READ_AUTHLOGS
- ROLE_BROKER_ADMIN (this is the super-user authority; a user with this can do anything)

For scopes, all admin tools (including the Broker Admin Console and Privacy Preferences app) will need to request the following scope when performing authentication.

- **urn:unboundid:broker:scope:default** -- The catch-all scope indicating the user has logged in successfully. It does not denote any particular authorities are held, but is necessary to allow an admin client to get a token.

Index