UnboundID® Identity Broker

Installation Guide

Version 5.1.0

UnboundID Corp

13809 Research Blvd., Suite 500

Austin, Texas 78750

Tel: +1 512.600.7700

Email: support@unboundid.com

Copyright

Table of Contents

Preface

The UnboundID Identity Broker Installation Guide provides procedures to install and configure an identity infrastructure.

About UnboundID

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

The UnboundID Platform provides the following:

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

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

About This Guide

This guide provides procedures to install and configure an Identity infrastructure, powered by the UnboundID product suite. The guide references products in the UnboundID product family including:

- UnboundID Identity Broker
- UnboundID Identity Data Store
- UnboundID Identity Proxy
- UnboundID Identity Data Sync Server
- UnboundID Identity Broker API
- UnboundID Server SDK

Additional documentation for each product is available.

Audience

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

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

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

Documentation

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

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

Chapter 1: Introduction

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

This section includes:

Identity Broker Overview

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

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

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

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

Identity Broker Features

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

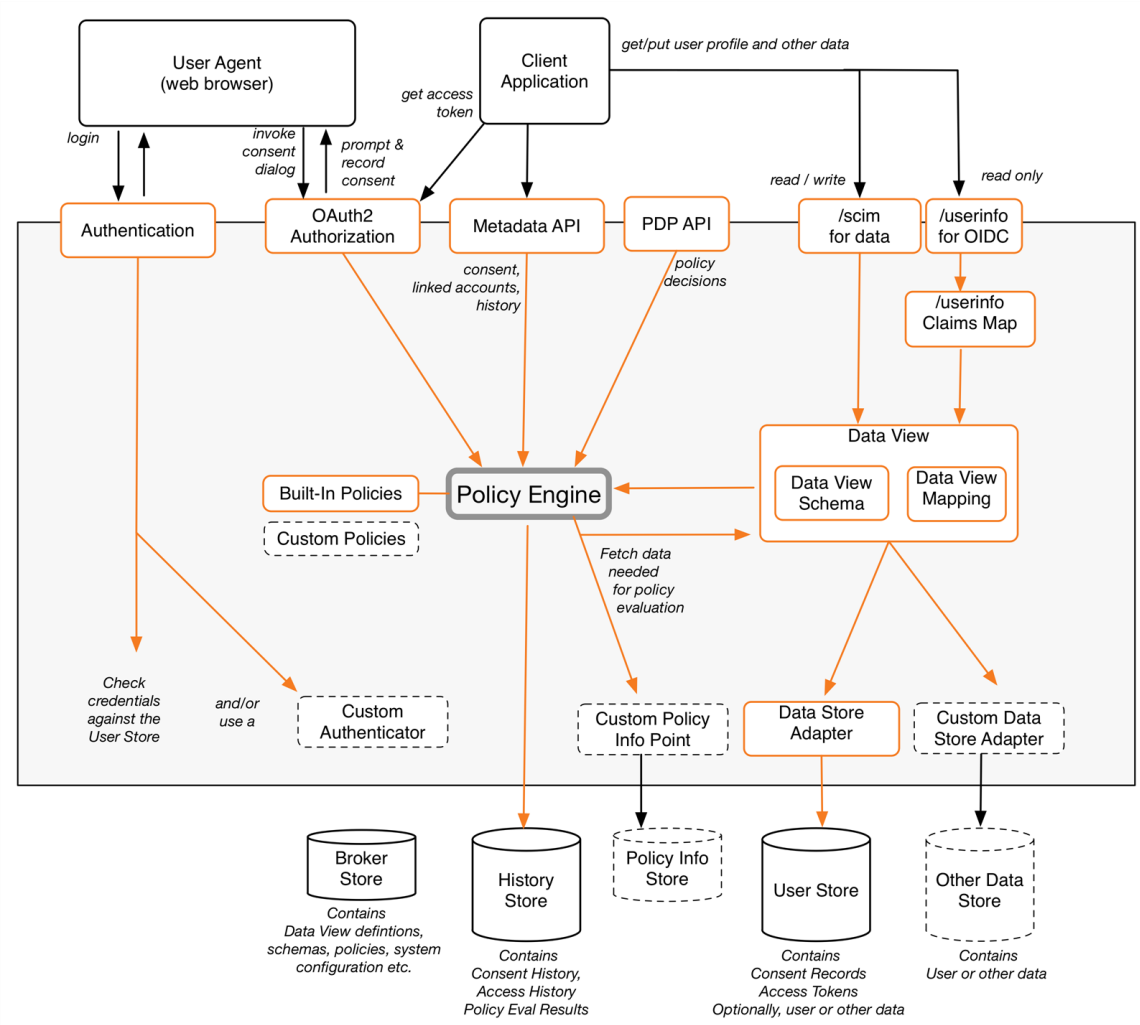
- **Support for multiple backend data stores**. The Identity Broker supports multiple data stores, with native support for the UnboundID Data Store and extension points for other data stores, such as relational databases. Applications can be written one time for access to the Identity Broker and receive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

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

Identity Broker 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. Backend data stores are configured as User Stores during installation. Data Views enable mapping resources from multiple User Stores into a unified view of a user profile.

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.

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.

Installation Considerations

Consider the following deployment-related issues prior to installing:

- **Determine the role of the Identity Broker Server**. The Identity Broker can serve as a resource and authorization server, or a resource server. The role of the server will determine the configuration requirements.
- **Determine the Identity Broker Store Topology**. The deployment determines where the Identity Broker stores its policies, Data View Schemas, and OAuth 2.0 tokens for each user.
- **Determine the Identity Broker and Broker Store load balancing and replication scenarios**. Multiple Identity Brokers can be installed for load balancing. Install one Identity Broker and use the clone feature to install additional Identity Brokers, or plan a scripted installation. Multiple Identity Brokers can use a single Broker Store. Make sure that the Broker Store has a backup or replication mechanism in place. See About the Broker Store and User Store.
- **Code required for Application and Resource Server**. The Identity Broker provides REST API endpoints for web, mobile, social and partner applications as well as resource server access to the OAuth 2.0 and policy services and the administrative tools. See the *UnboundID Identity Broker Client Developer Guide* for more information.

Chapter 2: System Requirements

The UnboundID Identity Broker requires few technical prerequisites and can be deployed in multiple configurations. The Identity Broker can be deployed on virtualized and/or commodity hardware, and monitored using the UnboundID Platform's built-in tools or through external tools connected with the API.

This section includes:

Installation Prerequisites

The following are required before installing the Identity Broker:

- Java 7
- Minimum of 2 GB RAM
- UnboundID Identity Data Store 4.5 or later

Note

If using the log history service, the amount of disk space required will depend on the chosen configuration. See the "Managing the Log History Service" section of the *UnboundID Identity Broker Administration Guide* for information about the service and configuration details.

Supported Platforms

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

Supported Storage Options

The Identity Broker can be deployed in a variety of topologies depending on the existing infrastructure. The following table lists the Identity Broker components that must reside on an Identity Data Store, or can reside on a third-party data store.

Summary of Storage Options

Store	Identity Data Store	Third-Party Directory or Database
Consumer (end user) Accounts	Yes	Yes
Broker Store	Yes	No
Administrator Accounts	Yes	Yes

Configuring File Descriptor Limits

Identity Broker allows for an unlimited number of connections by default, but is restricted by the file descriptor limit on the operating system. Many Linux distributions have a default file descriptor limit of 1024 per process, which may be too low to handle a large number of concurrent connections.

Set the maximum file descriptor limit per process to 65,535 on Linux systems.

To Set the File Descriptor Limit

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

```
$ ulimit -aH
```

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

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

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

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

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

```
$ ulimit -n
```

Setting the Maximum User Processes

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

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

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

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

Installing the dstat Utility on SuSE Linux

The `dstat` utility is used by the `collect-support-data` tool to gather support data. It can be obtained from the OpenSuSE project website. The following steps install the `dstat` utility on SuSE Enterprise Linux 11 SP2:

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

```
$ zypper addrepo
http://download.opensuse.org/repositories/server:/monitoring/SLE_11_SP2
Monitoring
```

3. Install the `dstat` utility:

```
$ zypper install dstat
```

Chapter 3: Installation

Identity Broker provides installation tools to quickly configure the server.

This section includes:

Installing the JDK

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

Solaris systems require both the 32-bit (installed first) and 64-bit versions. The 64-bit version of Java on Solaris relies on a number of files provided by the 32-bit installation.

About Encryption Keys

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

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

See Managing the Server Encryption Settings for more information.

Installing the Identity Data Store

The Identity Broker requires at least one installed Identity Data Store server. This provides the backend repository for the Broker Store, which contains the policy data, resources, actions, applications, and Data View Schemas (to enable mapping of attributes between the Identity Broker and one or more User Stores). A user store is also required by the Identity Broker, which can be an instance of the Data Store or an external user store. The Broker Store can reside with the User Store on a single Identity Data Store server, or multiple data stores can be installed.

See About the Broker Store and User Store for details about these components.

Note

All sensitive data in the user store are encrypted. When using the UnboundID Data Store as the user store, server-level encryption can be enabled as described in the "Encrypting Sensitive Data" section in the *UnboundID Identity Data Store Administration Guide*.

To Install the Identity Data Store

Follow this procedure to install a single Identity Data Store server. All configuration settings can be later modified through the `dsconfig` tool. The following information is needed during the installation:

- Server hostname
- LDAPS port

- Root DN and password
- Base DN
- Location of user entries

Perform the following steps to install the Identity Data Store:

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

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

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

   ```
   $ cd UnboundID-DS
   ```

4. Run the `setup` command.

   ```
   $ ./setup
   ```

5. Enter **yes** to agree to the license terms.
6. Enter the Directory Manager DN for the Data Store, or accept the default, (`cn=Directory Manager`). This account has full access privileges.
7. Enter a password for the root user DN, and confirm it.
8. Choose the communication option for SCIM and the Configuration API. HTTPS is recommended.
9. Enter the port to accept connections from HTTPS clients or press **Enter** to accept the default. The default may be different depending on the account privileges of the user installing. This port defines the URL port (such as `https://<hostname>:8443/`) required when installing Identity Broker.
10. Enter the port to accept connections from LDAP clients, or press **Enter** to accept the default.
11. Type **yes** to enable LDAPS, or press **Enter** to accept the default (no). When configuring the Identity Broker, the `create-initial-broker-config` tool assumes that this is enabled.
12. If enabling LDAPS, enter the port to accept connections, or press **Enter** to accept the default LDAPS port.
13. Type **yes** to enable StartTLS for encrypted communication, or press **Enter** to accept the default (no).
14. Select the certificate option for the server and provide the certificate location.

    ```
    Certificate server options:
    ```

```
   1)  Generate self-signed certificate (recommended for testing
       purposes only
   2)  Use an existing certificate located on a Java KeyStore (JKS)
   3)  Use an existing certificate located on a PKCS12 keystore
   4)  Use an existing certificate on a PKCS11 token
```

15. The server listens on all available network interfaces. To specify particular IP addresses that accept client connections, enter **yes** and then enter the IP addresses. To keep all interfaces available for connections, press **Enter** to accept the default (no).
16. Specify the base DN for the Identity Data Store repository, for example `dc=company,dc=com`.
17. Select an option to populate the database. If this data store will serve as a user store for the Identity Broker, it should be populated with users. If the **Leave the database empty** option is selected, an LDIF file with a base entry must be manually created at a later time. Use `ldapmodify` to add the entry to the Identity Data Store.
18. If this machine is dedicated to the Data Store, tune the JVM memory allocation to use the maximum amount of memory the **Aggressive** option). This ensures that communication with the Data Store is given the maximum amount of memory. Choose the best memory option for the system and press **Enter**.
19. Enter **yes** to automatically prime the database, or press **Enter** to accept the default (no).
20. To start the server after the configuration, press **Enter** for (yes).
21. Review the Setup Summary, and enter an option to accept the configuration, redo it, or cancel.

```
Setup Summary
=============
SCIM Web Services (SSL): https://<hostname>:443
Root User DN:         cn=Directory Manager
LDAP Listener Port:   1389
HTTP Listener Port:   disabled
Secure Access:        Enable SSL on LDAP Port 636
                      Enable SSL on HTTP Port 443
                      Create a new Self-Signed Certificate
Directory Data:       Create New Base DN dc=company,dc=com
                      Base DN Data: Import Automatically-Generated Data (2000 Entries)

The Identity Data Store will be started after configuration
What would you like to do?
```

```
   1)  Set up the server with the parameters above
   2)  Provide the setup parameters again
   3)  Cancel the setup
```

22. Choose the LDAP option to connect to the Data Store on the host.

```
>>>>> Specify LDAP connection parameters


 1)  LDAP
 2)  LDAP with SSL
```

23. Enter the Administrator user bind DN (directory manager), or press **Enter** to accept the default (`cn=Directory Manager`).
24. Enter and confirm a password for this account.

The Data Store configuration is displayed and the installation is complete.

Identity Broker Installation Tools

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

- The `setup` tool performs the initial tasks needed to start the Identity Broker server, including configuring JVM runtime settings and assigning listener ports for the Broker's REST services and web applications.
- The `create-initial-broker-config` tool continues after `setup` and enables initial system configuration. During the process, the `prepare-external-store` tool loads the Broker Store with an initial data set, including an administrative account. If specified, the configuration process calls the `sample-data-loader` tool, loads sample applications, OAuth 2 scopes, resources, user consent records, and authorization requests. Configuration can be written to a file to use for additional installations.
- The Broker Console interface or the `broker-admin` tool are used to define policies, attributes, and data resources for the system. The Broker Console interface enables all configuration that the `broker-admin` tool provides.
- Once the configuration is done, the `dsconfig` tool enables more granular configuration.

Installation Process and Files Installed

During the installation and configuration of the Identity Broker, there are opportunities to install sample data and prepare the system for immediate use after the installation is complete. For very advanced administrators, these steps can be scripted, or done manually with the `dsconfig` and `broker-admin` tools. For a simplified and interactive installation, use the integrated `setup` and `create-initial-config` tools.

One of the Identity Broker's key features is the ability to create Data Views, which rely on a SCIM schema to map attributes in a back-end data store to SCIM attributes or OpenID Connect resources. When specifying a Broker Store during the `create-initial-broker-config` process, the `broker-admin` script `install-data-view-mappings.broker-admin` is run. Data View mappings for a SCIM schema are created for the default User Store Adapter and User Data View. This enables an Identity Broker administrator to quickly map attributes from the selected user store to SCIM attributes or OpenID Connect resources in the Identity Broker Console. Additional user stores, Store Adapters, Data View Schemas, and Data Views can be created and configured at any time.

One of the final steps to configuring the Identity Broker is to write the configuration to the server and to a file. This activates all of the configuration settings entered and saves the configuration to a `dsconfig` batch file. The `dsconfig` tool can be used to further configure the server or configure additional Identity Brokers. The file `resource/install-oidc-objects.broker-admin` is parameterized and run. This file will:

- Create a User Data View.
- Create OpenID Connect scopes (profile, email, address, phone).
- Create claims maps.

The final steps of configuring the Identity Broker enable default policies and install sample data. This enables an Identity Broker administrator to use the Broker immediately. The default policies can be used as is, modified, or used as templates for additional policies. The XML files are imported into the Broker Store.

The installation enables installing sample policies as well as those required by the system:

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

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

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

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

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

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

If sample data is installed, the following are performed:

- Two users are created over SCIM: `sampleuser1` and `sampleuser2`.
- The `sample-data-loader` tool is run with the `install` subcommand. The newly created users serve as XACML resource owners.
- The `sample-data-loader` tool will create:
  - Tags, resources, trust levels, and scopes using the `broker-admin` tool.
  - The `consent-admin` tool is run with a batch file that adds READ access consents to the customer profiles for the newly installed applications.

See About the sample-data-loader Tool for details.

Installing the Identity Broker

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

- An administrative account for the Identity Broker.
- An available port for the Identity Broker to accept HTTPS connections from REST API clients. This port will be used by the Identity Broker's HTTPS Connection Handler.
- The web applications to install with this Identity Broker instance.
- An available port for the web applications' communication.
- An available port to accept LDAP client connections.
- Information related to the server's connection security, including the location of a keystore containing the server certificate, the nickname of that server certificate, and the location of a truststore.
- The network interfaces to be assigned to client communication. If specific interfaces are not assigned, all available interfaces are used.

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

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

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

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

   ```
   $ ./setup
   ```

5. Type **yes** to accept the terms of this license agreement.

6. The `setup` tool enables cloning a configuration by adding to an existing Identity Broker topology. For an initial installation, press **Enter** to accept the default (no).
7. Enter the fully qualified host name or IP address of the machine that hosts the Identity Broker, or press **Enter** to accept the default (local hostname).
8. Enter the Directory Manager account DN for the Identity Broker. This account has full access privileges. To accept the default (`cn=Directory Manager`), press **Enter**.
9. Enter and confirm a password for this account.
10. Enter the port for the Identity Broker REST APIs to accept HTTPS client connections.This port is used by the Identity Broker to respond data requests or OAuth 2.0 requests. Press **Enter** to accept the default.
11. To install the Identity Broker Console web application on this Identity Broker instance press **Enter** to accept the default (yes).
12. Enter an HTTPS port to be used for the Identity Broker Console, or press **Enter** to accept the default.
13. Enter the port to accept LDAP client connections, or press **Enter** to accept the default.
14. To enable LDAPS connections type **yes** and enter a port, or press **Enter** to accept the default (no). If defined, the Identity Broker uses this port to access the backend user store or Broker Store.
15. To enable StartTLS connections over regular LDAP connection type **yes**, or press **Enter** to accept the default (no).
16. For secure connections (SSL or LDAPS), enter the certificate option for this server.
17. By default, all network interfaces on this server are used to listen for client connections. Type **yes** to designate specific addresses on which the Identity Broker listens for client connections, or press **Enter** to accept the default (no).
18. If this machine is dedicated to the Identity Broker, tune the JVM memory to use the maximum amount of memory (the **Aggressive** option). If this system supports other applications, choose an appropriate option.
19. Press **Enter** (yes) to start the server when the configuration is applied.
20. Review the configuration options and press **Enter** to accept the default (set up the server).

```
Setup Summary
=============
Broker Web Apps Port:          1445
Root User DN:                  cn=Directory Manager
LDAP Listener Port:            1389
HTTPS Listener Port:           1443
```

```
Secure Access:                 Enable SSL on LDAP Port 443
                               Create a new Self-Signed Certificate
                               Generate default trust store

The Identity Broker will be started after configuration

What would you like to do?

    1)  Set up the server with the parameters above
    2)  Provide the setup parameters again
    3)  Cancel the setup
```

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

Configuring the Identity Broker

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

- At least one Identity Broker Data Store is installed to host the Broker Store, which will contain policy and configuration information. The Identity Broker Data Store can also be used as a user store, which will contain user data and consent information. Have the host name and communication port available.
- Any additional Identity Data Stores or Proxy Servers that act as user stores. Only UnboundID Data Stores can be configured with this tool. Other user stores must be configured outside of this process. Have the host names and communication ports available.
- Locations for this and any other Identity Brokers for failover.
- The LDAP search filter to locate user entries in each user store, such as `(objectClass=person)`.

After the initial setup and configuration, run the `dsconfig` tool later to make configuration adjustments.

Note

All of the configuration information in this procedure can be written to the `broker-cfg.dsconfig` file and used to install additional servers, or additional servers can be configured with the identical configuration. This file contains sensitive information and should be secured.

1. Press **Enter** (yes) to continue with `create-initial-broker-config`.
2. Define the physical location of the Identity Broker server. Locations, typically, refer to the city where the data center resides. This location will be used to define where the Broker Store is located. The Identity Broker and the Broker Store should be in the same location for best performance.

```
Create a location name for this Identity Broker: austin
```

3. To define failover locations for other Identity Broker servers, enter **yes**. Failover locations can be defined later when additional Identity Broker servers are installed or cloned. Locations entered here are used to select the location of the Broker Store later in this configuration. Press **Enter** to accept the default (no) until other Identity Brokers are defined.
4. Define the account and password used by the Identity Broker to communicate with any external store, or press **Enter** to accept the default (`cn=Broker User,cn=Root DNs,cn=config`). An external store can hold user store data and/or be the location of the Broker Store.
5. Specify the type of security that the Identity Broker uses when communicating with all external store instances, or press **Enter** to accept the default (SSL).
6. Enter the `host:port` configured for the first Identity Data Store. The connection is verified.
7. Select the location name for the Broker Store, or enter another location if not listed in the menu.
8. Specify the base DN where the Broker Store data will be located on the Identity Data Store server. Press **Enter** to accept the default (`ou=Identity Broker,dc=example,dc=com`) or select the second option to enter another base DN.
9. Enter an administrative account to be used by Identity Broker Console and `broker-admin` tool users, or press **Enter** to accept the default (admin). Enter and confirm a password for this account.
10. Confirm that the identified host should be prepared. This is required if installing sample data later in the install process. If additional servers will be added as backups to the Broker Store, select the **Yes, and all subsequent servers** option. This enables the identification of another server later in the configuration. The `prepare-external-store` tool can also be used to perform these tasked at a later time.

```
Would you like to prepare host:636 for access by the Identity Broker?

    1)  Yes
    2)  No
```

```
    3)  Yes, and all subsequent servers
    4)  No, and all subsequent servers
Enter choice [3]:
```

11. Create the Identity Broker root user `cn=Broker User,cn=Root DNs,cn=config` account on the Identity Data Store server, which enables server to server access. Administrators or users do not use this account. Press **Enter** to accept the default (yes).

```
Would you like to create or modify root user 'cn=Broker User,
cn=Root DNs,cn=config' so that it is available for this
Identity Broker? (yes / no) [yes]:
```

12. Enter the DN and password credentials needed to create the root user `cn=Broker User,cn=Root DNs,cn=config` account on the Identity Data Store. This is the root account created in the initial setup, such as default (`cn=Directory Manager`. The Identity Broker sets up the DN and tests that it can access the account. The Broker Schema and Policy Structure are also imported and verified.
13. If there are additional servers that will host the Broker Store data, enter their `host:port` for LDAP communication. If the option to prepare multiple servers was selected, the additional servers will be prepared with the same configuration that was just defined. If there are no additional servers to add, press **Enter** to continue.
14. If user data stores are ready to be configured (Identity Data Stores or Identity Proxy servers), press **Enter** for (yes). The user store will be configured with a default Store Adapter and Data View, which will enable mapping of resources in the user store to the Identity Broker.
15. Enter the `host:port` for the first Identity Data Store or Identity Proxy Server, or press **Enter** to accept the default.
16. Select an option to prepare the user store for access by the Identity Broker and press **Enter**.
17. If there are additional user data store locations, enter their `host:port`. If there are no additional servers to add, press **Enter** to continue.
18. Specify the base DN for locating user entries, such as `ou=people,dc=example,dc=com` and press **Enter**.
19. Create an LDAP search filter for this DN and press **Enter**.
20. The filter is validated against the DN. Press **Enter** (yes) to use these settings.
21. Review the configuration summary, and then press **Enter** to accept the default (w) to write the configuration to a `dsconfig` batch file. The configuration is written to `<server-root>/broker-cfg.dsconfig` . Certificate files are written to `external-server-certs.zip`.

```
>>>> Configuration Summary

  Admin Service URL:           https://<hostname>:1443/auth/api/v1
  OAuth2 Service URL:          https://<hostname>:1443/oauth
  Policy Service URL:          https://<hostname>:1443/pdp/v1
  User Metadata Service URL:   https://<hostname>:1443/metadata/v1

  OpenID Connect Service URL:  https://<hostname>:1443/userinfo
  SCIM Service URL:            https://<hostname>:1443/scim/Users

  Identity Broker Console:     https://<hostname>:1445/broker-console

  Identity Broker Location: austin

  Broker Store
    Base DN: ou=Identity Broker,dc=example,dc=com
    Identity Broker User DN: cn=Broker User,cn=Root DNs,cn=config
    Connection Security: SSL
    Servers: <hostname>:389

  User Store
    Base DN: ou=people,dc=example,dc=com
    Search Filter: (objectClass=inetOrgPerson)
    Identity Broker User DN: cn=Broker User,cn=Root DNs,cn=config
    Connection Security: SSL
    Servers: <hostname>:389

What would you like to do?

  b)  back
  q)  quit
  w)  write configuration file
```

22. Press **Enter** (w) to confirm that the configuration should be applied to this Identity Broker and written to the `broker-cfg.dsconfig` file.
23. Press **Enter** to confirm the configuration.
24. Install general-purpose policies that are ready for use or can be used as a starting point in configuring additional policies. Press **Enter** to accept the default (yes).

25. Select the option to load sample data so that the Identity Broker can be used immediately after setup and press **Enter**. If not, data can be added at a later time using the `sample-data-loader` tool.
26. Two sample applications can be installed with the Identity Broker to demonstrate how a client can access the Identity Broker's resources, and how end-users can view and manage consent to access those resources. To install these applications, select option (1).
27. This completes the initial configuration for the Identity Broker. Run the `bin/status` tool to see that the Identity Broker server is up and running.

The UnboundID Identity Broker and its web applications are installed. Start the Identity Broker Console, `https:<hostname>:<8445>/broker-console` to verify the connection.

Installing a Clone Identity Broker

An Identity Broker instance can be cloned to serve as an additional server. Cloning a server copies the original Identity Broker's local configuration and links the two configurations. Both Identity Brokers will share the same Broker Store and user stores.

For the installation process, the first Identity Broker is called the peer server. The new server is called the cloned server. Review To Install the Identity Broker for details about each option. Once the configuration is complete, the two servers are peers.

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

1. Unpack the zip distribution in a folder different from the peer Identity Broker.
2. Run the `./setup` command in the `<server-root>` directory of the cloned server.
3. Accept the licensing agreement.
4. Enter **yes** to add this server to an existing Identity Broker topology.
5. Enter the host name of the peer Identity Broker server from which the configuration will be copied.
6. Enter the port of the peer Identity Broker.
7. Choose the security communication to use to connect to the peer Identity Broker.
8. Enter the manager account DN and password for the peer Identity Broker, or press **Enter** to accept the default (`cn=Directory Manager`). The connection is verified.
9. Enter the fully-qualified host name or IP address of the local host (the cloned server).
10. Enter the HTTPS client connection port for the Identity Broker, or press **Enter** to accept the default.
11. Select the option to install the Identity Broker Console application, if desired.

12. Enter the HTTPS connection port for the Identity Broker applications, or press **Enter** to accept the default.
13. Enter the port on which the clone Identity Broker will accept connections from LDAP clients, or press **Enter** to accept the default.
14. To enable LDAPS, enter **yes**.
15. To enable StartTLS, enter **yes**.
16. To specify particular addresses on which the server will listen to client connections enter **yes**.
17. Enter **yes** to tune the JVM memory for performance. If yes, enter the amount of memory to allocate to the JVM.
18. Enter **no** to so that he server is not started after configuration.

Note

The encryption settings backend must be backed up from the instance being cloned, copied to the Identity Broker being installed, and restored before the server can be started.

19. Review the information for the configuration, and press **Enter** to set up the server with these parameters.
20. To write this configuration to a file, press **Enter** to accept the default (yes).
21. The clone is installed and configured based on the configuration settings of the peer.

After cloning the Identity Broker, the encryption settings backend must be backed up from the instance being cloned, copied to the new Identity Broker, and restored with the following commands:

```
backup –-backupDirectory /tmp/backup –-backendID encryption-settings
```

Copy `/tmp/backup` to the target server and then restore the backup to the cloned Identity Broker:

```
restore –-backupDirectory /tmp/backup
```

Planning a Scripted Install

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

When developing an installation script, it can be helpful to first install a reference Identity Broker using the interactive process. The results from the installation script can then be compared to the reference Identity Broker.

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

External store preparation:

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

Server configuration with `dsconfig`:

- A `broker-cfg.dsconfig` batch file is generated and loaded to define the Identity Broker's server configuration. This creates external server and load balancing algorithm configuration objects needed by the LDAP store adapter, configures important properties of the OAuth 2.0 Service, and sets initialization parameters needed by the Identity Broker Console application.

Broker Store configuration with `broker-admin`:

- The Broker Store is updated with the file `<server-root>/resource/install-oidc-objects.broker-admin`. This creates the default User Data View and adds scopes and UserInfo claims mappings needed for OpenID Connect support.
- The Broker Store is updated with the file `<server-root>/resource/install-data-view-mappings.broker-admin`. This creates default attribute mappings from the User Data View to the LDAP store adapter.
- Required XACML policies are loaded into the Broker Store.
- The default Consent Policy is linked to the OAuth Policy Sandbox.

Sample applications and data installation:

- Two test users are created and the `sample-data-loader` tool is run to generate sample applications, scopes, resources, consent records, and XACML authorization requests.
- The sample Sign In application is installed.
- The sample Profile Manager application is installed.

Certain actions are performed dynamically and require special logic in a scripted installation:

Client credentials are dynamically generated for the Identity Broker command-line tools and the Identity Broker Console application by `prepare-external-store` and stored in the file `<server-root>/tmp/create-initial-broker-config.props`.

- The client credentials for the command-line tools must be set in the OAuth 2.0 Service configuration object.

- The client credentials for the Identity Broker Console application must be set in the `Broker-Admin-Console` web application extension object.

A file path placeholder variable in the `<server-root>/resource/install-oidc-objects.broker-admin` must be expanded.

Scripted Installation Process

If a scripted installation is done without the use of the `create-initial-broker-config` tool, the process may look like this:

1. Set up and configure one or more Identity Data Stores. See To Install the Identity Data Store.
2. Run the Identity Broker `setup` tool on the server that will host the Identity Broker.
3. Run `prepare-external-store` for the stores.
4. The client ID and secrets for the Broker Console and command-line tools are stored in `<server-root>/tmp/create-initial-broker-config.props`. The following is an example of the file contents:

   ```
   client-secret=[command-line tools client secret]
   client-id=[command-line tools client ID]
   admin-console-client-secret=[Broker Console client secret]
   admin-console-client-id=[Broker Console client ID]
   ```

   The client credentials should be extracted from this file (using Perl or Unix tools such as `grep` and `cut`) and inserted into the dsconfig batch file. For example:

   ```
   $ dsconfig set-oauth-service-prop \
     --set "oauth-admin-client-id:[command-line tools client ID]" \
     --set "oauth-admin-client-secret:[command-line tools client secret]" \
     --set "id-token-issuer-name:broker.example.com
   ```

   ```
   $ dsconfig set-web-application-extension-prop \
     --extension-name Broker-Admin-Console \
     --set "oauth-admin-client-id:[Broker Console client ID]" \
     --set "oauth-admin-client-secret:[Broker Console client secret]
   ```

5. Load the dsconfig batch file:

   ```
   $ dsconfig --no prompt --batch-file broker-cfg.dsconfig
   ```

6. Substitute the Identity Broker server root path for the placeholder string `$SERVER_ROOT` in `<server-root>/resource/install-oidc-objects.broker-admin`.
7. Run the `broker-admin`command to load OpenID Connect objects:

   ```
   $ broker-admin --no-prompt --batch-file <server-root>/resource/install-oidc-objects.broker-admin
   ```

8. Run the `broker-admin`command to load Data View mappings:

```
$ broker-admin --no-prompt --batch-file <server-root>/resource/install-
data-view-mappings.broker-admin.
```

9. Import required policies with the `broker-admin import-policy` tool:

```
$ broker-admin import-policy \
  --enable --overwrite --xmlFile "<server-
root>/resource/AdminAccess.xml" \
  --name "Admin API Policy"
```

```
$ broker-admin import-policy \
--enable --overwrite --xmlFile "<server-
root>/resource/ConsentPolicy.xml" \
--name "Consent Policy"
```

```
$ broker-admin import-policy \
  --enable --overwrite --xmlFile "<server-
root>/resource/resource/GovernanceTagPolicy.xml" \
  --name "Tag Policy"
```

```
$ broker-admin import-policy \
  --enable --overwrite --xmlFile "<server-
root>/resource/resource/TrustLevelPolicy.xml" \
  --name "Trust Level Policy"
```

```
$ broker-admin import-policy \
  --enable --overwrite --xmlFile "<server-
root>/resource/UserCreateAndUpdatePolicy.xml" \
  --name "User Create and Update Policy"
```

```
$ broker-admin import-policy \
  --enable --overwrite --xmlFile "<server-
root>/resource/OwnedResourcePolicy.xml" \
  --name "Owned Resource Access Policy”
```

10. Configure the OAuth Policy Sandbox with the following command:

```
$ broker-admin set-policy-sandbox-prop \
  —id OAuthConsent@
  —set “policyIds:name=Consent Policy”
```

11. If desired, add two users to the User Store and run the `sample-data-loader` tool.
12. If desired, unzip `<server-root>/samples/sign-in.zip` and install the Sign In sample application according to its README file.
13. If desired, unzip `<server-root>/samples/profile-manager.zip` and install the Profile Manager sample application according to its README file.
14. Confirm that the Identity Broker Console application is configured properly by logging in as the admin user. Do the same for the other sample applications, if they were installed.
15. When finished, delete the `<server-root>/tmp/create-initial-broker-config.props` file.

Note

The redirect URI that the Identity Broker Console uses (defined in the server configuration as a property of the `Broker-Admin-Console` web application extension) must match one of the redirect URIs that the Identity Broker expects (defined as a property of the "UnboundID Broker Admin Console" application in the Broker Store). See Changing the Identity Broker Console Redirect URI for information.

To Install the Identity Broker with an Existing Truststore

By default, the `setup` command configures your certificates and installs the keystore and truststore in the `config` directory (i.e., `config/keystore` and `config/truststore`). If you want to use an existing keystore and truststore in a different path, you can run the `setup` tool, then run the `create-initial-broker-config` separately. The following procedures run `setup` from the command-line in non-interactive mode. You can also run it interactively, but do not run the `create-initial-broker-config` tool during the same session.

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

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

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

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

Chapter 4: Management

The Identity Broker provides server management tools needed to run basic functions, such as stop, start, uninstall, and others. The tools are located in the server root directory or in the `bin` directory of the server.

This section includes the following:

Run the Identity Broker

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

To Run the Identity Broker

On the command line, run the following command.

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

To Run the Identity Broker in the Foreground

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

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

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

Stop the Identity Broker

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

To Stop the Identity Broker

Use the `bin/stop-broker` tool to shut down the server.

```
$ bin/stop-broker
```

Schedule a Server Shutdown

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

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

To Run an In-Core Restart

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

```
$bin/stop-broker --task --restart --hostname 127.0.0.1 \
--bindDN uid=admin,dc=example,dc=com --bindPassword password
```

Uninstalling the Identity Broker

The Identity Broker provides an `uninstall` tool to remove the components from the system.

To Uninstall the Identity Broker

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

   ```
   $ ./uninstall
   ```

1. Select the option to remove all components or select the components to be removed.

   ```
   Do you want to remove all components or select the components to remove?

       1)  Remove all components
       2)  Select the components to be removed

       q)  quit

   Enter choice [1]: 2
   ```

2. To selected components, enter **yes** when prompted.

   ```
   Remove Server Libraries and Administrative Tools? (yes / no) [yes]: yes
   Remove Log Files? (yes / no) [yes]: no
   Remove Configuration and Schema Files? (yes / no) [yes]: yes
   Remove Backup Files Contained in bak Directory? (yes / no) [yes]: no
   Remove LDIF Export Files Contained in ldif Directory? (yes / no) [yes]: no

   The files will be permanently deleted, are you sure you want to continue? (yes / no)
   [yes]:
   ```

3. Manually delete any remaining files or directories.

Updating the Identity Broker, Broker Store, and User Stores

Updating an Identity Broker deployment includes:

- Updating the Broker Store using the `prepare-external-store` tool, which updates the Broker Store's version, and may add schema elements and configuration elements needed by the new Identity Broker release.
- Updating each configured User Store using the `prepare-external-store` tool.
- Updating the Identity Broker server(s) using the `update` tool, which updates the Identity Broker itself.

The Identity Broker expects Broker Store data to be at least as current as the Identity Broker server version, and an Identity Broker server will not start using a prior version of the Broker Store. Therefore, the Broker Store must be updated prior to updating the Identity Broker server.

Note

Upgrade is only supported for releases 5.0.0 or 5.0.1 to this release.

Perform the following steps to update the Broker Store:

1. Before updating the existing Broker Store, use the `backup` tool to backup the Broker Store data.
2. Gather information about the existing Broker Store including the base DN and connection information to the Data Store.
3. Obtain a new Identity Broker installation package and unzip the file in a temporary directory on the same host as the Identity Broker instance to be updated.
4. Update the Broker Store by using the `--update` option with the `prepare-external-store` tool. If there are multiple Data Stores replicating the Broker Store data, it is only necessary to run the command on one of the servers:

```
$ prepare-external-store --update --isBrokerStore \
  --brokerStoreBaseDN <baseDN> \
  --hostname <host> \
  --port <port> \
  <--useSSL>
  --bindDN <directory-manager-bind-DN> \
  --bindPassword <directory-manager-bind-password>
```

5. The Broker Store is now current with the new Identity Broker package.
6. Run `prepare-external-store --update` on each configured User Store.
7. Run the `update` tool from the temporary directory with the new Identity Broker installation package. For example:

```
$ /path/to/new-broker-files/update --serverRoot /path/to/original-
broker-files
```

8. Test the new Identity Broker installation. There may be other manual, post-update steps necessary. See the "Upgrade Considerations" section of the Identity Broker release notes for information specific to this release.
9. Update any other Identity Broker instances.

Identity Broker Files and Folders

Once you have unzipped the Identity Broker distribution file, the following folders and command-line utilities are available.

Layout of the Identity Broker Folders

Directories/Files/Tools	Description
LICENSE.txt	Licensing agreement for the Identity Broker.
README	README file that describes the steps to set up and start the Identity Broker.
bak	Stores the physical backup files used with the backup command-line tool.
bat	Stores Windows-based command-line tools for the Identity Data Store.
broker-cfg.txt	Stores the configuration history for the Identity Broker. Appears after you have configured the Identity Broker.
classes	Stores any external classes for server extensions.
collector	Stores collector files.
config	Stores the configuration files and the directories for messages, schema, tools, and updates.
docs	Provides the release notes, Configuration Reference file and a basic Getting Started Guide (HTML).
import-tmp	Stores temporary imported items.
ldif	Stores any LDIF files that you may have created or imported.
legal-notices	Stores any legal notices for dependent software used with the Identity Broker.
lib	Stores any scripts, jar, and library files needed for the server and its extensions.
locks	Stores any lock files in the backends.
logs	Stores log files for the Identity Broker.
metrics	Stores files for the UnboundID Metrics Engine.
resource	Stores the MIB files for SNMP.
revert-update	The revert-update tool for UNIX/Linux systems.
revert-update.bat	The revert-update tool for Windows systems.
setup	The setup tool for UNIX/Linux systems.
setup.bat	The setup tool for Windows systems.

Directories/Files/Tools	Description
tmp	Temp directory.
unboundid_logo.png	UnboundID logo
uninstall	The uninstall tool for UNIX/Linux systems.
uninstall.bat	The uninstall tool for Windows systems.
update	The update tool for UNIX/Linux systems.
update.bat	The update tool for Windows systems.
webapps	Stores the war files for reference implementations.

Chapter 5: Configuration

This chapter describes how to configure optional components that may be needed to customize the installation with Identity Broker tools and configuration.

Sections include the following:

Identity Broker Configuration Data

The Identity Broker's non-user data consists of data in the server configuration and data in the Broker Store. Generally, data in the server configuration define an individual Identity Broker instance in its capacity as a network server and are primarily of interest to system administrators. Data in the Broker Store define an entire Identity Broker deployment in its capacity as an identity and authorization service and are of primary interest to identity service administrators.

The server configuration is stored in an LDIF-based backend under the `cn=config base DN`. It can be accessed using the LDAP protocol and is managed by the `dsconfig` tool or the Identity Broker Console. Much of the data stored in the server configuration are settings that are used by that particular Identity Broker instance. Examples of data in the server configuration include connection handler listener ports and log publisher settings. When an Identity Broker is set up from a peer, its server configuration is cloned to the new Identity Broker, and the two configurations are linked such that changes to the configuration are applied to both Identity Broker servers by default. See Installing a Clone Identity Broker.

The Broker Store is a logical data store layer backed by one or more UnboundID Identity Data Stores and (optionally) one or more UnboundID Data Proxy servers. Data stored in the Broker Store is accessed over HTTP using the Identity Broker's Admin API and is managed using the `broker-admin` tool, the Identity Broker Admin Console, or a custom Admin API client. The data stored in the Broker Store are higher-level operational data or business logic used by the Identity Broker service as a whole. When multiple Identity Broker servers exist in a topology, the Broker Store data is always shared by all servers. Examples of Broker Store data are policies, applications, scopes, and resources.

Identity Broker Configuration Tools

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

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

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

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

All Identity Broker Tools

Available Identity Broker tools are:

Identity Broker Tools

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

Tool	Description
base64	Encode raw data using the base64 algorithm or decode base64-encoded data back to its raw representation.
broker-admin	Invoke administrative operations over the Identity Broker REST API.
collect-support-data	Collect and package system information useful in troubleshooting problems. The information is packaged as a ZIP archive that can be sent to a technical support representative.
consent-admin	Manage a resource owner consent over the Identity Broker REST API. Consent is authorized by a resource owner to allow access to resources by an application.
config-diff	Generate a summary of the configuration changes in a local or remote server instance. The tool can be used to compare configuration settings when troubleshooting issues, or when verifying configuration settings on new servers.
create-initial-broker-config	Create an initial Identity Broker configuration.
create-rc-script	Create a Run Control (RC) script that can be used to start, stop, and restart the Identity Broker on Unix-based systems.
dsconfig	View and edit the Identity Broker configuration.
dsframework	Manage administrative server groups or the global administrative user accounts that are used to configure servers within server groups.
dsjavaproperties	Configure the JVM arguments used to run the Identity Broker and its associated tools. Before launching the command, edit the properties file located in `config/java.properties` to specify the desired JVM arguments and the `JAVA_HOME` environment variable.
encryption-settings	Manage the server encryption settings database.
evaluate-policy	Request a policy decision from the Identity Broker.
ldapmodify	Perform LDAP modify, add, delete, and modify DN operations in the Identity Broker.
ldappasswordmodify	Perform LDAP password modify operations in the Identity Broker.
ldapsearch	Perform LDAP search operations in the Identity Broker.
ldif-diff	Compare the contents of two LDIF files, the output being an LDIF file needed to bring the source file in sync with the target.
ldifmodify	Apply a set of modify, add, and delete operations against data in an LDIF file.
list-backends	List the backends and base DNs configured in the Identity Broker.
manage-extension	Install or update extension bundles. An extension bundle is a package of extension (s) that utilize the Server SDK to extend the functionality of the Identity Broker. Any added extensions require a server re-start.
oauth2-request	Performs OAuth 2.0 requests on the Identity Broker. This tool can be used to test OAuth 2.0 functions of the Identity Broker, and to manage OAuth 2.0 tokens on behalf of registered applications.
prepare-external-store	Prepares the external data stores for the Identity Broker. This is run as part of the `create-initial-broker-config` tool during installation. This tool creates the broker user account, sets the correct password, and configures the account with required privileges. It will also install the necessary schema required by the Identity Broker. This tool can also be used to update (with the `--update` option) an external Broker Store or a data store schema.
remove-defunct-server	Removes a permanently unavailable Identity Broker after it has been removed from

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

Using the Web Console for Server Configuration

All UnboundID servers can be managed with the UnboundID Web Console, or the `dsconfig` command-line tool. The console provides configuration and schema management functionality in addition to monitoring and server information. Like the `dsconfig` configuration tool, all changes made using the console are recorded in `logs/config-audit.log`. In addition, anytime a configuration is made to the system, the configuration backend is automatically updated and saved as gzip-compressed files. Changes can be accessed in the `config/archived-configs` folder. The console must be deployed in a servlet container that supports the servlet API 2.5 or later. An installation using Apache Tomcat is described below for illustration purposes.

Note

The console supports JBoss 7.1.1 or later. Refer to the JBoss Compatibility section in the `WEB-INF/web.xml` file for specific configuration steps.

This console is separate from the Identity Broker Console, which is used to configure the identity environment.

Install the Web Console

1. Download and install Apache Tomcat.
2. Set the appropriate environment variables. The `setclasspath.sh` and `catalina.sh` files are in the tomcat `/bin` directory.

   ```
   $ echo "BASEDIR=/path/to/tomcat" >> setclasspath.sh
   $ echo "CATALINA_HOME=/path/to/tomcat" >> catalina.sh
   ```

3. Download and unzip the console ZIP file, `UnboundID-broker-web-console-<version>.zip` on the local host. The following files are available:

   ```
   3RD-PARTY-LICENSE.TXT
   LICENSE.TXT
   ```

```
README
dsconsole.war
```

4. Create a `dsconsole` directory in `apache-tomcat-<version>/webapps/`.
5. Copy the `dsconsole.war` file to `apache-tomcat-<version>/webapps/dsconsole`. If the servlet is running and auto-deploy is enabled, copy the `.war` file to the `/webapps` directory and it will install in the directory.

```
$ mkdir apache-tomcat-<version>/webapps/dsconsole
$ cp dsconsole.war apache-tomcat-<version>/webapps/dsconsole
```

6. Go to the `apache-tomcat-<version>/webapps/dsconsole` directory to extract the contents of the console. The jar command is included with the JDK.

```
$ cd apache-tomcat-<version>/webapps/dsconsole
$ jar xvf dsconsole.war
```

7. Optional. Edit the `WEB-INF/web.xml` file to point to the correct Identity Broker instance. The parameters in the `web.xml` file appear between as comments. Uncomment `<!-- parameter -->` the needed parameters. For example, specify the server or servers that the console uses to authenticate with the following parameters:

```
<context-param>
  <param-name>ldap-servers</param-name>
  <param-value>localhost:389</param-value>
</context-param>
```

   If the `ldap-servers` parameter is not specified, the web console displays a form field for the user to enter the server host and port. If configured, the console also uses this server to "discover" other servers in the topology, making them available for monitoring and management in the console.

8. Optional. With the default configuration, Tomcat sessions will time out after 30 minutes of inactivity. This can be changed on a servlet container wide basis by editing `apache-tomcat-<version>/conf/web.xml`, and updating the value of this configuration parameter:

```
<session-config>
  <session-timeout>120</session-timeout>
</session-config>
```

   The session expires after the specified number of minutes. Changes to this setting might not take effect until the servlet container is restarted, so consider changing the value before starting the server for the first time.

9. Optional. To remove sensitive information from console error messages, such as LDAP and server information, edit `apache-tomcat-<version>/conf/web.xml`, and change the value of this configuration parameter to `false`:

```
<context-param>
  <param-name>detailedErrorMessages<param-name>
```

```
  <param-value>false<param-value>
<context-param>
```

When set to `false`, the console will display a generic error page with server information removed. Server logs will still contain detailed error information.

10. Start the Identity Broker if it is not already running, and then start the console using the `apache-tomcat-<version>/bin/startup.sh script`. Use `shutdown.sh` to stop the servlet container. (On Microsoft Windows, use `startup.bat` and `shutdown.bat`.) The JAVA_HOME environment variable must be set to specify the location of the Java installation to run the server.

```
$ env JAVA_HOME=/ds/java bin/startup.sh
Using CATALINA_BASE:    /apache-tomcat-<version>
Using CATALINA_HOME:    /apache-tomcat-<version>
Using CATALINA_TMPDIR:  /apache-tomcat-<version>/temp
Using JRE_HOME:         /ds/java
```

Open a browser to `http://<hostname:8080>/dsconsole`. By default, Tomcat listens on port 8080 for HTTP requests. If the Identity Broker is restarted, log out of the current console session and then log back in.

Log into the Web Console

To log into the console, either use a DN (for example, `cn=Directory Manager`) or provide the name of an administrator stored under `cn=admin` data.

1. Navigate to to the server root directory.

```
$ cd UnboundID-DS
```

2. Start the Identity Broker.

```
$ bin/start-ds
```

3. Start the Apache Tomcat application server.

```
$ /apache-tomcat-<version>/bin/startup.sh
```

4. In a browser, open `http://<hostname:8080>/dsconsole/`.
5. Type the root user DN (or any authorized administrator user name) and password, and then click Login.
6. On the Console, click **Configuration**.
7. View the Configuration menu. By default, the console displays the Basic object type properties. You can change the complexity level of the object types using the **Object Types** drop-down list.

Configure the Web Console

The console uses a `web.xml` descriptor file for its configuration and deployment settings. In addition to configuring one or more primary servers, the application time out, and a generic web page in the previous procedure, the security and truststore settings for the console can also be set.

To Configure SSL for the Primary Console Server

Configure the console to communicate with servers over SSL or StartTLS. See the previous section on how to specify one or more primary servers.

1. Open the `dsconsole/WEB-INF/web.xml` file in a text editor to specify the type of communication to authenticate. First, remove the comment tags (<!-- and -->) in the security section.
2. Specify none, ssl, or starttls for the type of security that you are using to communicate with the Identity Data Store.

```
<context-param>
  <param-name>security</param-name>
  <param-value>ssl</param-value>
</context-param>
```

3. Save the file.

To Configure a Truststore for the Console

For SSL and StartTLS communication, specify the truststore and its password (or password file) in the `web.xml` file. If no truststore is specified, all server certificates will be blindly trusted.

1. Open the `dsconsole/WEB-INF/web.xml` file in a text editor and remove the comment tags (`<!--` and `-->`) in the truststore section.
2. Specify the path to your truststore.

```
<context-param>
  <param-name>trustStore</param-name>
  <param-value>/path/to/truststore</param-value>
</context-param>
```

3. Specify the password or the path to the password pin file.

```
<context-param>
  <param-name>trustStorePassword</param-name>
  <param-value>password</param-value>
</context-param>

<context-param>
  <param-name>trustStorePasswordFile</param-name>
  <param-value>/path/to/truststore/pin/file</param-value>
</context-param>
```

4. Save the file.

Upgrade the Web Console

Upgrade the console by moving the `web.xml` file to another location, unpacking the latest console distribution, and then replacing the newly deployed `web.xml` file with the previous file.

1. Shut down the console and servlet container.
2. Move the current `webapps/dsconsole/WEBINF/web.xml` file to another location.
3. Download and deploy the latest version of the console. See Install the Web Console.
4. Perform a `diff` between the previous and newer version of the `web.xml` file to determine changes that should be applied to the new file. Make any necessary changes to the new file.
5. Start the servlet container.

Uninstall the Web Console

Uninstall the existing console by removing the `webapps/dsconsole` directory.

1. Close the console, and shut down the servlet container. (On Microsoft Windows, use `shutdown.bat`.)

```
$ apache-tomcat-<version>/bin/shutdown.sh
```

2. Remove the `webapps/dsconsole` directory.

```
$ rm -rf webapps/dsconsole
```

3. Restart the servlet container instance if necessary. If no other applications are installed in the servlet instance, the entire servlet installation can be removed.

Using the dsconfig tool

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

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

```
$ bin/dsconfig
```

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

```
$ dsconfig --bindDN uid=admin,dc=company,dc=com \
  --bindPassword password \
```

```
  --no-prompt \
  --batch-file </path/to/config-batch.txt>
```

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

To Run the dsconfig Tool

Initial configuration for the Identity Broker was defined during setup. Use this tool to refine or change the initial configuration. The tool requires the Identity Broker server connection information.

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

   ```
   $ bin/dsconfig
   ```

2. Enter the Identity Broker hostname or IP address and press **Enter**.
3. Specify the option to connect to the Identity Broker and press **Enter**.
4. Enter the connection port, or press **Enter** to confirm the default (1389).
5. Enter the administrator user bind DN, or press **Enter** to confirm the default (`cn=Directory Manager`).
6. Enter the password for this account and press **Enter**. The Identity Broker configuration main menu is displayed.

   ```
   >>>> UnboundID Identity Broker configuration console main menu
   What do you want to configure?

   1)  Alarm Manager                    14) Load Balancing Algorithm
   2)  Alert Handler                    15) Location
   3)  Broker Store                     16) Log History Service
   4)  Connection Handler               17) Log Publisher
   5)  External Server                  18) Log Retention Policy
   6)  Gauge                            19) Log Rotation Policy
   7)  Gauge Data Source                20) Oauth Service
   8)  HTTP Authentication Scheme       21) Policy Service
   9)  HTTP Servlet Cross Origin Policy 22) Server Affinity Provider
   10) HTTP Servlet Extension           23) Store Adapter
   11) HTTP User Authenticator          24) Velocity Context Provider
   12) Identity Provider Adapter        25) Velocity Template Loader
   13) LDAP Health Check                26) Web Application Extension


   o)   'Standard' objects are shown - change this
   q)   quit
   ```

7. Choose the configuration option and press **Enter**.

Using the Configuration API

UnboundID servers provide a Configuration API, which may be useful in situations where using LDAP to update the server configuration is not possible. The API features a REST-ful design and uses JSON as a text exchange format, so all request headers should allow the application/json content type.

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

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

The API is made available on the HTTPS Connection handler's `host:port` in the `/config` context. Due to the potentially sensitive nature of the server's configuration, use a secure connection handler such as the HTTPS Connection Handler, for hosting the Configuration extension.

Authentication and Authorization

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

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

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

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

Relationship Between the Configuration API and the dsconfig Tool

The Configuration API is designed to mirror the `dsconfig` tool, using the same names and formats for configuration object types. Like the `dsconfig` tool, all configuration updates made through the API are recorded in `logs/config-audit.log`.

Like `dsconfig`, the Configuration API can request specific properties to be returned in object retrieval and object list operations. For list operations, the query parameter `property` can be specified to indicate specific properties to include. Unless specifically requested, a small

subset of default properties is returned. The value '*' can be used as the value of the `property` query parameter to request all properties be returned in an object listing.

Object retrieval operations, by default, return all properties for an object. The `property` query parameter can be used to limit the set of properties returned to a specified set.

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

HTTP Method	Description	Related dsconfig Example

GET	Lists the properties of an object when used with a path representing an object, such as `/config/global-configuration` or `/config/backends/userRoot`. Can also list instances objects when used with a path representing a parent relation, such as `/config/backends`.	`get-backend-prop`
		`list-backends`
		`get-global-configuration-prop`

POST	Creates a new instance of an object when used with a relation parent path, such as `config/backends`.	`create-backend`
PATCH	Updates the properties of an existing object when used with a path representing an object, such as `/config/backends/userRoot`.	`set-backend-prop` `set-global-configuration-prop`
DELETE	Deletes an existing object when used with a path representing an object, such as `/config/backends/userRoot`.	`delete-backend`

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

Note

Use of the PUT method for object creation or modification is not supported.

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

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

API Paths

The Configuration API is available under the `/config` path. A full listing of supported sub-paths is available by accessing the base `/config` path. The following is a sample list:

```
{
  "status" : "OK",
  "message" : "You have accessed the configuration base URL. You can use the following pa
ths to access the configuration.",
  "paths" : [
    "/access-control-handler",
    "/account-status-notification-handlers",
    "/account-status-notification-handlers/{handlerName}",
    "/alarm-manager",
...
```

The schema's `paths` element enumerates all available sub-paths. The path `/config/backends` in the example can be used to get a listing of existing backends as well as create new ones. A path containing an object name like `/config/backends/{backendName}`, where `{backendName}` corresponds to an existing backend (such as `userRoot`) may be used to obtain an object's properties, update the properties, or delete the object.

Some paths reflect hierarchical relationships between objects. For example, properties of a local DB VLV index for the `userRoot` backend are available using a path like `/config/backends/userRoot/local-db-indexes/uid`. Some paths represent singleton objects, which have properties but cannot be deleted nor created. These paths can be differentiated from others by their singular, rather than plural, relation name (for example `global-configuration`).

Updating Properties

Configuration object properties can be modified with an HTTP PATCH request on a path representing an object or singleton object. Request bodies must consist of a JSON object enumerating a property operation along with a set of property/value pairs, or a list of properties. Operations correspond with and behave the same as the options used in `dsconfig set-{object}-prop` subcommands. Multiple properties can be specified for each operation.

The following sample patch request updates a backend, setting the description property, updating the multi-valued `je-property`, and resetting the `id2children-index-entry-limit` to its original setting:

```
{
  "set" : [ { "description" : "The user backend" } ],
  "remove" : [ {
    "je-property" : "je.env.backgroundReadLimit=3000" } ],
  "add" : [ {
    "je-property" : "je.cleaner.adjustUtilization=false" }, {
    "je-property" : "je.env.backgroundReadLimit=0" } ],
  "reset" : [ "id2children-index-entry-limit" ]
}
```

Administrative Actions

Updating a property may require an administrative action before the change can take effect. If so, the server will return `200 Success`, and the request body will contain an encoded set of actions to be performed. For example, changing the `je-property` of a backend will result in the following:

```
{
  "required-actions" : [ {
  "property" : "je-property",
  "type" : "component-restart",
  "synopsis" : "In order for this modification to take effect, the component must be rest
arted, either by disabling and re-enabling it, or by restarting the server"
 } ]
}
```

Configuration API Responses

Clients of the API should examine the HTTP response code in order to determine the success or failure of a request. The following are response codes and their meanings:

Response Code	Description	Response Body
200 Success	The requested operation succeeded, with the response body containing the requested data, or further actions that are required.	List of objects, or object properties, administrative actions.
201 Created	The requested operation to create a new object succeed. A link to the newly created object is sent in the Location response header field.	None.
204 No Content	The requested operation succeeded and no further information has been provided, such as in the case of a DELETE operation.	None.
400 Bad Request	The request contents are incorrectly formatted or a request is made for an invalid API version.	Error summary and optional message.
401 Unauthorized	User authentication is required. Some user agents such as browsers may respond by prompting for credentials. If the request had specified credentials in an Authorization header, they are invalid.	None.
403 Forbidden	The requested operation is forbidden either because the user does not have sufficient privileges or some other constraint such as an object is edit-only and cannot be deleted.	None.
404 Not Found	The requested path does not refer to an existing object or object relation.	Error summary and optional message.
406 Not Acceptable	The request is such that the Accept header does not indicate that JSON is an acceptable format for a response.	None.
409 Conflict	The requested operation could not be performed due to the current state of the configuration. For example, an attempt was made to create an object that already exists or an attempt was made to delete an object that is referred to by another object.	Error summary and optional message.
500 Server Error	The server encountered an unexpected error. Please report server errors to customer support.	Error summary and optional message.

An application that uses the Configuration API should limit dependencies on particular text appearing in error message content. These messages may change, and their presence may depend on server configuration. Use the HTTP return code and the context of the request to create a client error message. The following is an example encoded error message:

```
{
  status: "Not Found"
  message: "Relation `bad-relation' does not exist"
}
```

The Configuration extension has an `omit-error-message-details` property that suppresses the message in error responses, preventing the server from inadvertently revealing sensitive information. Set this property as follows:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name "Configuration" \
  --set omit-error-message-details:true
```

About the Broker Store and User Store

During the Identity Broker configuration, one or more UnboundID Identity Data Store or Proxy instances are identified to store policy definitions, application registry, and identity service configuration, and may also serve as a user store. The User Store is a repository of actual user data. These data include user attributes such as names, email addresses, and preferences, as well as user-specific metadata needed by the Identity Broker, such as authorization codes and access tokens. The User Store may actually be composed of one or more physical data stores, and they may be of varying types. For example, some user data may be stored in an LDAP directory server while other user data may be stored in a relational database or a document database. Data views, described further in the next section, provide a consistent abstracted view of types of users in a User Store.

The user store is used to store user profile data, as well as metadata for authorization and consent history. Some installations may have existing user stores that require the installation of a separate metadata store, due to corporate policy or access restrictions. If an existing user store is not allowed to write Identity Broker metadata for user access, see Configuring a Separate Metadata Store for details.

The Broker Store is a repository of operational data and business logic that serve the Identity Broker's function as an authorization service, such as:

- Resource objects that represent end user identity attributes.
- Applications that represent entities that require access to end user resources.
- Actions that represent the kind of access desired by applications.
- Policies that define the logic that determines authorization decisions.

Note

If there are multiple Identity Data Stores hosting the Broker Store, all instances should be configured to replicate the data beneath the Broker Store base DN. See the *UnboundID Identity Data Store Administration Guide* for replication information.

About Data Views

A user is represented in the Identity Broker as a Data View. A Data View represents user data that are potentially aggregated from disparate sources in the User Store as a consistent, unified profile. Applications that consume services provided by the Identity Broker, such as the SCIM service, the Metadata API, or the policy decision point, address users and user attributes

in terms of a Data View and do not need to be concerned with details specific to the physical data stores that back the User Store.

A Data View is defined in terms of a collection of attributes that a user may have expressed as a SCIM 1.1 schema. All user attributes are named according to SCIM schema notation. For example, a username may be `urn:scim:schemas:core:1.0:userName`. All attributes are represented in the Broker Store as resources. For more information about configuring and managing data views, see the *UnboundID Identity Broker Administration Guide*.

The following are required to enable Data Views:

User Stores – The Identity Broker requires at least one existing user store, which can be an Identity Data Store, an existing LDAP directory, or other third-party directory. When a user store is defined through the `create-initial-broker-config` tool, a Store Adapter and Store Attribute Map are created. These enable mapping of the native schema attributes (attributes native to the user store) to attributes that will be defined by a Data View Schema and surfaced in a Data View.

Data View Schema – A SCIM schema must be created in JSON format and imported into the Identity Broker Console. The schema will contain a number of SCIM attributes that should be mapped to attributes in Identity Data Stores or third-party user stores. The schema can represent a single SCIM resource, such as User or Group, which can contain one or more attributes. The schema name and the Data View created for it must match exactly.

The Identity Broker provides a sample SCIM schema that can be used as a template in `<server-root>/resource/example-starter-schema`.

Data View – A Data View is created in the Identity Broker Console application or with the `broker-admin` tool, and is associated with a Data View Schema of the same name. Attributes from the associated Data View schema are mapped to the attributes from the associated user store or stores.

About Store Adapters

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

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

When a store adapter is added to the Identity Broker's server configuration, a correlation attribute can be defined for Data Views that are backed by multiple store adapters. The correlation attribute defines an attribute for each store adapter that is used to uniquely identify

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

Creating custom store adapters requires the UnboundID Server SDK. The Server SDK documentation describes the API available to store adapter implementations. It is not usually necessary to implement all features of the API for each store adapter. For example, if user login credentials are stored in a specific data store, then only the store adapter for that data store needs to support authentication. User metadata may be stored in another data store, so only the corresponding store adapter needs to support user metadata. See Server Extensions for information.

About the LDAP Store Adapter

The LDAP Store Adapter is a generic implementation of the store adapter, enabling it to interface with any vendor's LDAP server, such as the Identity Data Store or Proxy Server, or Oracle DSEE.

LDAP Store Adapter instances are configured using the `dsconfig` tool. Configuration parameters include:

- `correlation-attribute-urn` – Determines the native SCIM attribute to use as the correlation between resources when multiple Store Adapters are configured in a Data View.
- `modifies-as-creates` – Determines if this store adapter will process a modify operation as a create operation if the target resource does not exist.
- `load-balancing-algorithm` – Specifies the load balancing algorithm.
- `structural-ldap-object-class` – Determines the LDAP object class exposed by this adapter.
- `auxiliary-ldap-object-class` – Optionally specifies one or more auxiliary LDAP object classes to expose.
- `include-base-DN` – Determines the base DNs for the branches of the LDAP directory that can be accessed by this LDAP Store Adapter.
- `include-filter` – Determines set of LDAP filters that define the LDAP entries that should be included in this LDAP Store Adapter.
- `scim-id-attribute` – Determines the LDAP attribute to use as the SCIM ID when returning entries in SCIM format.
- `include-operational-attribute` – Determines the set of operational LDAP attributes to include in the native SCIM schema that is provided by this LDAP Store Adapter.
- `create-DN-pattern` – Determines the template to use for the DN when creating new entries.

About User Metadata

User metadata is any user-specific data that is created and maintained by the Identity Broker for its own purposes. This includes consent records, OAuth 2.0 authorization codes, and access tokens. For any given data view, at least one store adapter must support storing user metadata.

User metadata is divided into three categories. Each metadata type is multivalued.

- **Small metadata** – This includes consent records, OAuth 2.0 authorization codes, access tokens, and refresh tokens. This data is generally small and is accessed frequently.
- **Large metadata** – This includes data such as consent history. This data can be large and is accessed less frequently.
- **Indexed metadata** – This includes data such as links to user accounts at external identity providers. This data should be indexed to allow for efficient equality searches.

The format of user metadata is entirely private to the Identity Broker, and should only be accessed and manipulated through the Identity Broker's Metadata API. All other uses are unsupported. Store adapter implementations need not be concerned with the format of user metadata but should consider the expected sizing, indexing, and access patterns.

The following should be considered when configuring a store adapter and determining where user metadata is stored:

- Not all store adapters need to support storing user metadata, but at least one store adapter in an Identity Broker environment must.
- For every Data View, at least one store adapter must handle user metadata.
- Metadata can be stored in multiple store adapters, for redundancy purposes. User stores should be configured to support load balancing and failover.

A Data View stores the metadata in all store adapters that support it. For those store adapters that do not need to store metadata, the `user-metadata-attribute` and `user-large-metadata-attribute` properties can be disabled using the `dsconfig` tool.

In the case of an LDAP Store Adapter, both the small and large metadata attributes are multi-valued, binary attributes. The `LDAPStoreAdapter` configuration object has the following defaults:

```
user-metadata-attribute: id-broker-user-metadata
user-large-metadata-attribute: id-broker-user-large-metadata
```

An example user entry with `user-metadata-attribute` and `user-large-metadata-attribute` attributes might look like this:

```
dn: uid=jsmith,ou=people,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetorgperson
```

```
uid: jsmith
cn: John Smith
givenName: John
sn: Smith
userPassword: {SSHA}rcYNUGsFQXdM27VS+s/Uat/ydb5wruBmR2avwg==
id-broker-user-metadata: dGVzdGluZzEyMyR0ZXN0aW5nNDU2
id-broker-user-metadata: dGVzdGluZzAxMiR0ZXN0aW5nMjEw
id-broker-user-metadata: JHRlc3Rpbmc3ODk=
id-broker-user-large-metadata: YXNkZ2YgYXNkIGFzZGYgYXNkZ2Fkc2ZaGFk
ZmhhZHNmaGFkc2ZoYWRzZmhhc2RmZ2FzZGYgYXNkZ2FzZGdoYXNkZmhhc2RnYXNkZ2
hhc2ZkaGFzZGZnYXNkZ2ggc0RGSEFTREZIQVNERkhzZGdzREcgQVNEIEdBU0RHIEFE
U0ZIIEFTUkhBU0RGQVNEIDIgQVNERkcgQVNEIFRRV1RHU0dBU0RH
```

Configuring a Separate Metadata Store

The Identity Broker stores metadata, such as tokens, authorization codes, consents, and access history, on a per-user basis. By default, all user profile attributes and user metadata are stored in the user store. If necessary, the user metadata and consent history can be stored in a metadata store, separate from the user profile attributes.

This can be useful if installing the Identity Broker with an existing user store that is read-only, or restricted. In this scenario, the Identity Broker will read attributes from the exiting user store and store information about consent history and transactions in the metadata store. A correlation attribute is used by the Data View to correlate entries from multiple store adapters to the same user identity. See Configuring Custom Store Adapters for more information.

Preparing to Configure a Metadata Store

Before configuring the Identity Broker to use a metadata store, gather the following information:

- Configuration details for the Data View that will use the metadata store. For example, the name of the Data View and the store adapters used by the Data View.
- The correlation attribute to be used by all store adapters belonging to the Data View.
- The connection information to the data stores that will be used by the metadata store.

About the Correlation Attribute

When a Data View is linked to multiple store adapters, a correlation attribute must be defined for each store adapter to identify user entries within each backend data store. The correlation attribute must refer to an attribute from the store adapter's native schema. It can be a different attribute for each store adapter associated with a Data View. However, the value of the attribute must be the same across all adapters for a particular user. The value should be a primary key, such as the username, which corresponds to the `urn:scim:schemas:core:1.0:userName` attribute in the default User schema and `urn:unboundid:schemas:scim:ldap:1.0:uid` in an LDAP store adapter's native schema. Another possible choice is the unique ID, which corresponds to the `urn:scim:schemas:core:1.0:id` attribute in the default User schema.

Example: Configuring an LDAP Metadata Store

In this example, an UnboundID Data Store is used as a metadata store, and it will be added to an existing User Data View that uses another UnboundID Data Store as its User Store. See Installing the Data Store for details.

For this example, user entries are assumed to already reside in the `ou=people,dc=example,dc=com` base DN of the user store. The existing user entries are managed by the `UserStoreAdapter`. Each user entry in the user store will have a corresponding entry in the metadata store, but they will be created as they are needed. The metadata store is managed by the `MetadataStoreAdapter`.

This example uses the user's unique ID as a correlation attribute.

The following table provides an overview of the configuration values that will be used throughout this example.

Data View	`User`
User Store Adapter	`UserStoreAdapter`
User Store LBA	`User Store LBA`
User Store Base DN	`ou=people,dc=example,dc=com`
User Store Credentials	`cn=Broker User,cn=Root DNs,cn=config`
User Store Correlation Attribute	`urn:scim:schemas:core:1.0:id`
Metadata Store Adapter	`MetadataStoreAdapter`
Metadata Store LBA	`Metadata Store LBA`
Metadata Store Base DN	`ou=people,dc=example,dc=com`
Metadata Store Credentials	`cn=Broker User,cn=Root DNs,cn=config`
Metadata LDAP Object Class	`exampleIdentityBrokerUserMetadata`
Metadata Entry Filter	`(objectClass=exampleIdentityBrokerUserMetadata)`
Metadata Store Correlation Attribute	`example-broker-metadata-id`
Metadata Store User Metadata Attribute	`ds-broker-user-metadata`
Metadata Store User Large Metadata Attribute	`ds-broker-user-large-metadata`
Metadata Store Create DN Pattern	`example-broker-metadata-id={example-broker-metadata-id},ou=people,dc=example,dc=com`

The example scenario will change if a third-party Data Store and custom store adapter are used, but the general principles will apply.

Preparing the LDAP Data Store

The UnboundID Data Store as the metadata store is updated so that the schema defines an object class for storing user metadata, and the attribute used for correlating user entries from the user store to metadata entries on the metadata store is indexed.

1. Create a custom schema file:

```
dn: cn=schema
objectclass: top
objectclass: ldapSubentry
objectclass: subschema
cn: schema
attributeTypes: ( example-broker-metadata-id-oid NAME 'example-broker-
metadata-id'
   SYNTAX 1.3.6.1.1.16.1
   EQUALITY uuidMatch ORDERING uuidOrderingMatch
   SINGLE-VALUE X-ORIGIN 'user defined' )
objectClasses: ( example-broker-user-metadata-oid NAME
'exampleIdentityBrokerUserMetadata'
   DESC 'Container for example Identity Broker user metadata'
   SUP top STRUCTURAL MUST (example-broker-metadata-id )
   X-ORIGIN 'user defined' )
```

This schema file defines the custom object class "`exampleIdentityBrokerUserMetadata`," which requires the LDAP attribute "`example-broker-metadata-id`." The `example-broker-metadata-id` attribute uses the UUID attribute syntax, with object ID "`1.3.6.1.1.16.1`." This syntax is chosen because these attributes will contain SCIM ID values, which the Broker represents as UUIDs. The attribute syntax depends on the correlation attribute chosen.

The actual metadata is stored as operational attributes of the metadata entries, which are added to the server with the `prepare-external-store` tool in step 5.

2. Copy the schema file to `<server-root>/config/schema/` as `99-broker-metadata-store.ldif`.
3. Restart the Data Store to activate the new schema.
4. Create the `ou=People,dc=example,dc=com` base DN in the metadata store, if it does not already exist.
5. Run the `prepare-external-store` command against the metadata store with the `--isUserStore` option. Though this data store will not be used as a user store, this will create a `cn=Broker User` login account needed for use by the Identity Broker. For example:

```
$ prepare-external-store
  --hostname <hostname>
  --port <port> \
  --bindDN "cn=Directory Manager" \
  --bindPassword <root DN password> \
  --isUserStore \
  --userStoreBaseDN ou=people,dc=example,dc=com
```

6. Create an equality index for the `example-broker-metadata-id` attribute.

```
$ dsconfig create-local-db-index \
  --index-name example-broker-metadata-id \
```

```
  --backend-name userRoot \
  --set index-type:equality
```

7. Stop the Data Store and rebuild the index.

```
$ rebuild-index \
  --baseDN "ou=people,dc=example,dc=com" \
  --index example-broker-metadata-id
```

8. Restart the Data Store. The data store is now ready to be used by the Identity Broker.

Configuring the Store Adapter

An LDAP store adapter is created to reference the LDAP Data Store that was previously configured. This store adapter is added to the configuration for the User data view, and the existing user store adapter's configuration is updated.

1. Create an external server entry on the Identity Broker to represent the metadata store Data Store.

```
$ dsconfig create-external-server
  --server-name MetadataStoreDS1 \
  --type unboundid-ds \
  --set server-host-name:<hostname> \
  --set server-port <port> \
  --set location:<location> \
  --set bind-dn "cn=Broker User,cn=Root DNs,cn=config" \
  --set password:<password> \
  --set authorization-method:none
```

2. Create a load-balancing algorithm for the metadata store.

```
$ dsconfig create-load-balancing-algorithm \
  --algorithm-name "Metadata Store LBA" \
  --type failover --set enabled:true \
  --set backend-server:MetadataStoreDS1
```

3. Create a store adapter for the metadata store.

```
$ dsconfig create-store-adapter
  --adapter-name MetadataStoreAdapter \
  --type ldap \
  --set correlation-attribute-
urn:urn:unboundid:schemas:scim:ldap:1.0:example-broker-metadata-id \
  --set modifies-as-creates:true \
  --set include-ldap-objectclass:ExampleIdentityBrokerUserMetadata \
  --set include-base-dn:ou=People,dc=example,dc=com \
  --set user-metadata-attribute:ds-broker-user-metadata \
  --set user-large-metadata-attribute:ds-broker-user-large-metadata \
  --set "load-balancing-algorithm:Metadata Store LBA" \
  --set create-dn-pattern:example-broker-metadata-id={example-broker-
metadata-id},ou=people,dc=example,dc=com
```

The correlation attribute referenced by this value is used to store the user's SCIM ID.

When `modifies-as-creates` is set to true, the store adapter will create an entry instead of failing if the data view receives a modification request for an entry that does not already exist. This enables metadata entries corresponding to entries on the user store to be created on the fly in the metadata store. The `create-dn-pattern` property defines a template that the store adapter uses to name new entries in the metadata store.

4. Add the metadata store adapter to the User data view.

```
$ dsconfig set-data-view-prop
  --view-name User \
  --add store-adapter:MetadataStoreAdapter
```

5. Update the existing `UserStoreAdapter`. This store adapter is configured by default to store user metadata, so that configuration will be removed.

```
$ dsconfig set-store-adapter-prop
  --adapter-name UserStoreAdapter \
  --set correlation-attribute-urn:urn:scim:schemas:core:1.0:id \
  --remove user-metadata-attribute:ds-broker-user-metadata \
  --remove user-large-metadata-attribute:ds-broker-user-large-metadata
```

The User data view needs to be re-initialized before the changes can take effect. Either restart the Identity Broker server, or disable then enable the User data view.

Configuring the Data View

Modify the User Data View attribute mappings so that the correlation attribute has a mapping for the metadata store adapter. In the Data View, the `id` attribute of the common schema is mapped to the `example-broker-metadata-id` attribute of the metadata store adapter. This establishes the correlation between the metadata entry and the corresponding user entry.

```
$ broker-admin set-dataview-mapping
  --dataview User \
  --adapter MetadataStoreAdapter \
  --commonURN urn:scim:schemas:core:1.0:id \
  --nativeURN urn:unboundid:schemas:scim:ldap:1.0:example-broker-metadata-id \
  --readable --writable --indexed
```

The metadata store is now ready. Test the configuration by performing an authorization with the Sign-In Sample application, included with the Identity Broker. See the *UnboundID Application Developer Guide* for information about the Sign-In Sample application.

Configuring Store Adapters

The Identity Broker comes with an LDAP store adapter that can be used to interface with backend data stores. Data Views, configured in the Identity Broker Console, can then be used to map the attributes in on or more data stores. This enables a unified view of data across multiple stores.

Third-party adapters can be created for other data stores with the Server SDK available in the `unboundid-server-sdk-<version>.zip` package.

Configuring a custom store adapter includes the following steps:

1. Create a store adapter.
2. Store it in the `/extensions` directory of the Identity Broker.
3. Create a Data View schema.
4. Map Store Adapter(s) and Data Views using the Identity Broker Console.

Example Store Adapter

The Server SDK provides an example implementation of a third-party store adapter. View the example and associated Javadocs in the Server SDK `docs/example-html/ExampleStoreAdapter.java.html` directory.

`ExampleStoreAdapter.java` is an implementation of a flat-file JSON store adapter, which stores the SCIM user data in JSON. At startup, all resources are loaded from the `json-file-path` parameter (`resource/user-database.json`). The example uses an in-memory hash map of SCIM resources mapped to their SCIM ID.

The example provides full operations plus filterable search support for add, update, and deletes. The example will perform a full-file rewrite on every change, because the file format is a serialized list of `Resources<BaseResource>`. The code example does not support sorting or resource versioning.

Creating a JDBC Store Adapter

The Server SDK provides an example implementation of a JDBC store adapter. The example provides full operations plus search support for add, update, and deletes and persists it to the `SCIM_RESOURCES` table. View the example and associated Javadocs in the `docs/example-html/ExampleJDBCStoreAdapter.java.html` directory.

`ExampleJDBCStoreAdapter.java` shows how to implement a single-table JDBC store adapter with generic SQL support. The adapter stores users in Java jdbc format, which enables mirroring attributes on an RDBMS server. The example code depends on an Apache Derby 10.10.1.1 jdbc driver jar that must be copied into the server's `lib` directory. The default input parameters are:

- `jdbc-driver-class = org.apache.derby.jdbc.EmbeddedDriver`
- `jdbc-url = jdbc:derby:storeadapter`

At startup, the code auto-initializes by looking for a sentinel file in the `init-sql-schema-path` property, which has a default value of `resource/example-jdbc-store-adapter/.example-jdbc-schema-created`. If the file does not exist, the database will create a table with a `;create=true` URL and populate it with the core user schema from the `create-scim-table.sql` table as follows:

```
CREATE TABLE SCIM_RESOURCES {
```

```
  ID                   VARCHAR(44) NOT NULL PRIMARY KEY,
  EXTERNALID           VARCHAR(64),
  META                 LONG VARCHAR,
  USERNAME             VARCHAR(32),
  NAME                 VARCHAR(32),
  FAMILYNAME           VARCHAR(32),
  GIVENNAME            VARCHAR(32),
  MIDDLENAME           VARCHAR(32),
  HONORIFICPREFIX      VARCHAR(16),
  HONORIFICSUFFIX      VARCHAR(16),
  DISPLAYNAME          VARCHAR(32),
  NICKNAME             VARCHAR(32),
  PROFILEURL           VARCHAR(255),
  TITLE                VARCHAR(32),
  PREFERREDLANGUAGE    VARCHAR(8),
  LOCALE               VARCHAR(8),
  TIMEZONE             VARCHAR(32),
  ACTIVE               BOOLEAN,
  PASSWORD             VARCHAR(128),
  EMAILS               LONG VARCHAR,
  ADDRESSES            LONG VARCHAR,
  PHOTOS               LONG VARCHAR,
  GROUPS               LONG VARCHAR,
  ENTITLEMENTS         VARCHAR(255),
  ROLES                VARCHAR(255),
  x509CERTIFICATES     VARCHAR(4096) FOR BIT DATA,
  WEBSITE              VARCHAR(255),
  EMAILVERIFIED        BOOLEAN,
  GENDER               VARCHAR(16),
  BIRTHDATE            DATE,
  PHONENUMBERVERIFIED BOOLEAN,
  JSON                 LONG VARCHAR NOT NULL
}
```

Extend or modify the schema by editing the `create-scim-table.sql` file.

Multi-valued attributes require a persistence mechanism, such as Spring Hibernate, so the full JSON serialized object is stored in a JSON attribute.

The SQL statements are inline but could be placed in a properties file for customization without recompilation.

If necessary, the `storeadapter` sub-directory in the `resource/example-jdbc-store-adapter`directory can be deleted and recreated.

Building the Extension

Build the JDBC store adapter by following the instructions in the Server SDK package for building an extension:

1. On the server where the adapter is configured, run the following command to create a directory where the adapter can be built:

   ```
   $ mkdir -p src/com/unboundid/directory/sdk/examples
   ```

2. Copy the example store adapter to the new directory:

```
$ cp docs/example-src/ExampleJDBCStoreAdapter.java
src/com/unboundid/directory/sdk/examples
```

3. Edit the `extension.properties` file to set values for the properties used to specify the name, version, and vendor information for the extension bundle.
4. Run the `build.sh` shell script (or `build.bat` batch file on Windows systems) to build and package the extension.

Installing the Extension

After the extension is built, perform the following to install it on the Identity Broker server:

1. On the Identity Broker server, run the following command:

```
$ ./bin/manage-extension \
  --install unboundid-server-sdk-
<version>/build/com.example.ExampleJDBC-1.0.zip
```

2. Downloaded the latest Derby driver `derby-10.10.2.0.jar` and copy it to the Identity Broker `/lib` directory.
3. Run the following command:

```
$ ./bin/dsconfig create-store-adapter \
  --adapter-name ExampleJDBC \
  --type third-party \
  --set extension-
class:com.unboundid.directory.sdk.examples.ExampleJDBCStoreAdapter
```

Account Recovery Configuration in the User Store

End users can recover Identity Broker account information or reset a password, if an Identity Data Store is configured as the primary User Store and one-time passwords (OTP) are enabled.

On the Identity Broker server, configuration for account recovery and new account registration are enabled by configuring the OAuth HTTP Servlet Extension with the `dsconfig` tool. See the *UnboundID Identity Broker Administration Guide* for details.

Configuration on the Identity Data Store requires creating and enabling the OTP mechanism and defining the delivery mechanism for reset tokens, as follows:

1. Create and enable the OTP delivery mechanism:

```
$ dsconfig create-otp-delivery-mechanism \
  --mechanism-name "Email OTP Delivery Mechanism" \
  --type email --set enabled:true \
  --set 'sender-address:do-not-reply@example.com'
```

2. Define the email server to deliver reset tokens:

```
$ dsconfig create-external-server \
  --server-name "Example.com SMTP" \
  --type smtp --set server-host-name:smtp.example.com
```

```
$ dsconfig set-global-configuration-prop \
  --set "smtp-server:Example.com SMTP"
```

3. Create and enable the extended operations handlers to generate and send reset tokens:

```
$ dsconfig create-extended-operation-handler \
  --handler-name "Single Use Tokens" \
  --type single-use-tokens --set enabled:true \
  --set "password-generator:One-Time Password Generator" \
  --set "default-otp-delivery-mechanism:Email OTP Delivery Mechanism"
```

```
$ dsconfig create-extended-operation-handler \
  --handler-name "Deliver Password Reset Token" \
  --type deliver-password-reset-token --set enabled:true \
  --set "password-generator:One-Time Password Generator" \
  --set "default-token-delivery-mechanism:Email OTP Delivery Mechanism"
```

Managing Server Encryption Settings

The server encryption settings database is managed by the `encryption-settings` command-line tool. The keys stored for the server are used to encrypt tokens, authorization codes, account linking codes, and external identity provider tokens. Encryption settings definitions can be created, listed, exported and imported. Help and examples are available with the following command:

```
$ bin/encryption-settings --help
```

Information about the cipher algorithms and transformations available for use is located in the *Java Cryptography Architecture Reference Guide* and *Standard Algorithm Name Documentation* available at http://download.oracle.com/javase/6/docs/technotes/guides/security.

System Alarms, Alerts, and Gauges

UnboundID servers provide tools to monitor and manage the health of the system. The Identity Broker Server provides delivery mechanisms (handlers) for administrative alerts using JMX or SNMP, in addition to standard error logging. All can be configured with the `dsconfig` tool.

Alerts and alarms reflect state changes within the server that may be of interest to a user or monitoring service. An alarm represents a stateful condition of the server or a resource that may indicate a problem, such as low disk space or external server unavailability. A gauge defines a set of threshold values with a specified severity that, when crossed, cause the server to enter or exit an alarm state. Gauges are used for monitoring continuous values like CPU load or free disk space (Numeric Gauge), or an enumerated set of values such as 'server available' or ‘server unavailable’ (Indicator Gauge). Gauges generate alarms, when the gauge’s severity changes due to changes in the monitored value. Like alerts, alarms have severity (NORMAL, WARNING, MINOR, MAJOR, CRITICAL), name, and message. Alarms will always have a

Condition property, and may have a Specific Problem or Resource property. If surfaced through SNMP, a Probable Cause property and Alarm Type property are also listed. Alarms can be configured to generate alerts when the alarm's severity changes.

There are two alert types supported by the server - standard and alarm-specific. The server constantly monitors for conditions that may attention by administrators, such as low disk space. For this condition, the standard alert is `low-disk-space-warning`, and the alarm-specific alert is `alarm-warning`. The server can be configured to generate alarm-specific alerts instead of, or in addition to, standard alerts. By default, standard alerts are generated for conditions internally monitored by the server. However, gauges can only generate alarm-alerts.

The server installs gauges for CPU, disk, and memory usage that can be cloned or configured through the `dsconfig` tool. Existing gauges can be tailored to fit each environment by adjusting the update interval and threshold values. Configuration of system gauges determines the criteria by which alarms are triggered. The Stats Logger can be used to view historical information about the value and severity of all system gauges.

The server is compliant with the International Telecommunication Union CCITT Recommendation X.733 (1992) standard for generating and clearing alarms. If configured, entering or exiting an alarm state can result in one or more alerts. An alarm state is exited when the condition no longer applies. An `alarm_cleared` alert type is generated by the system when an alarm's severity changes from a non-normal severity to any other severity. An `alarm_cleared` alert will correlate to a previous alarm when Condition and Resource property are the same. The Alarm Manager, which governs the actions performed when an alarm state is entered, is configurable through the `dsconfig` tool.

Like the Alerts Backend, which stores information in `cn=alerts`, the Alarm Backend stores information within the `cn=alarms` backend. Unlike alerts, alarm thresholds have a state over time that can change in severity and be cleared when a monitored value returns to normal. Alarms can be viewed with the `status` tool.

As with other alert types, alert handlers can be configured to manage the alerts generated by alarms. A complete listing of system alerts, alarms, and their severity is available in `<server-root>/docs/admin-alerts-list.csv`.

Alert Handlers

Alert notifications can be sent to administrators when significant problems or events occur during processing, such as problems during server startup or shutdown. The Identity Broker Server provides a number of alert handler implementations configured with the `dsconfig` tool, including:

- **Error Log Alert Handler** – Sends administrative alerts to the configured server error logger(s).
- **JMX Alert Handler** – Sends administrative alerts to clients using the Java Management Extensions (JMX) protocol. The server uses JMX for monitoring entries and requires that the JMX connection handler be enabled.

- **SNMP Alert Handler** – Sends administrative alerts to clients using the Simple Network Monitoring Protocol (SNMP). The server must have an SNMP agent capable of communicating via SNMP 2c.

If needed, the Server SDK can be used to implement additional, third-party alert handlers.

Test Alarms and Alerts

After gauges, alarms, and alert handlers are configured, verify that the server takes the appropriate action when an alarm state changes by manually increasing the severity of a gauge. Alarms and alerts can be verified with the `status` tool.

Perform the following steps to test alarms and alerts:

1. Configure a gauge with `dsconfig` and set the `override-severity` property to `critical`. The following example uses the CPU Usage (Percent) gauge.

```
$ dsconfig set-gauge-prop \
  --gauge-name "CPU Usage (Percent)" \
  --set override-severity:critical
```

2. Run the `status` tool to verify that an alarm was generated with corresponding alerts. The `status` tool provides a summary of the server's current state with key metrics and a list of recent alerts and alarms. The sample output has been shortened to show just the alarms and alerts information.

```
$ bin/status
```

```
                          --- Administrative Alerts ---
Severity : Time        : Message
---------:-------------:-----------------------------------------------------
Error    : 11/Aug/2014 : Alarm [CPU Usage (Percent). Gauge CPU Usage (Percent)
         : 15:41:00    : for Host System Recent CPU and Memory has
         :  -0500      : a current value of '18.583333333333332'.
         :             : The severity is currently OVERRIDDEN in the
         :             : Gauge's configuration to 'CRITICAL'.
         :             : The actual severity is: The severity is
         :             : currently 'NORMAL', having assumed this severity
         :             : Mon Aug 11 15:41:00 CDT 2014. If CPU use is high,
         :             : check the server's current workload and make any
         :             : needed adjustments. Reducing the load on the system
         :             : will lead to better response times.
         :             : Resource='Host System']
         :             : raised with critical severity
Shown are alerts of severity [Info,Warning,Error,Fatal] from the past 48 hours
Use the --maxAlerts and/or --alertSeverity options to filter this list
```

```
                           --- Alarms ---
Severity : Severity   : Condition : Resource    : Details
         : Start Time :           :             :
---------:------------:-----------:-------------:---------------------------
Critical : 11/Aug/2014: CPU Usage : Host System : Gauge CPU Usage (Percent) for
         : 15:41:00   : (Percent) :             : Host System
         :  -0500     :           :             : has a current value of
         :            :           :             : '18.785714285714285'.
```

```
        :           :           :             : The severity is currently
        :           :           :             : 'CRITICAL', having assumed
        :           :           :             : this severity Mon Aug 11
        :           :           :             : 15:49:00 CDT 2014. If CPU use
        :           :           :             : is high, check the server's
        :           :           :             : current workload and make any
        :           :           :             : needed adjustments. Reducing
        :           :           :             : the load on the system will
        :           :           :             : lead to better response times

Shown are alarms of severity [Warning,Minor,Major,Critical]
Use the --alarmSeverity option to filter this list
```

Server SDK Extensions

Custom server extensions can be created with the UnboundID® Server SDK. Extension bundles are installed from a .zip archive or a file system directory. Use the `manage-extension` tool to install or update any extension that is packaged using the extension bundle format. It opens and loads the extension bundle, confirms the correct extension to install, stops the server if necessary, copies the bundle to the server install root, and then restarts the server.

Note

The `manage-extension` tool must be used with Java extensions packaged using the extension bundle format. For more information, see the "Building and Deploying Java-Based Extensions" section of the Server SDK documentation.

The UnboundID Server SDK enables creating extensions for the Identity Data Store, Identity Proxy, Metrics Engine, Identity Broker, and Identity Data Sync servers. Cross-product extensions include:

- Access Loggers
- Alert Handlers
- Error Loggers
- Key Manager Providers
- Monitor Providers
- Trust Manager Providers
- OAuth Token Handlers
- Manage Extension Plugins

Extensions for the Identity Broker include:

- Policy Information Provider
- Store Adapter

About the OAuth Service

OpenID Connect, built on the OAuth 2.0 standard, is an identity layer that enables applications to authenticate end users without performing the authentication themselves. It also enables end-user identity data to be shared between interested parties with the end-users' consent. It provides two primary mechanisms for doing this:

- ID tokens. ID tokens are compact objects which provide information about authentication events.
- The UserInfo endpoint. This is a bearer token-protected REST endpoint which provides attributes ("claims") about a specific identity.

The OAuth2 implementation uses the Spring Security OAuth Framework, providing the necessary interfaces to develop an OAuth2 client application. After the Identity Broker is installed, the OAuth service can be configured with the `dsconfig` tool. The following are configuration options:

```
>>>> Configure the properties of the OAuth Service

Property                                  Value(s)
-----------------------------------------------------------------------------
1)   authorization-code-validity-duration  1 m
2)   access-token-validity-duration        12 h
3)   refresh-token-validity-duration       4 w 2 d
4)   reuse-refresh-tokens                  true
5)   user-approval-page-url                /view/oauth/approve
6)   error-page-url                        /view/oauth/error
7)   id-token-validity-duration            15 m
8)   id-token-issuer-name                  server.com
9)   signing-algorithm                     hs256

?)   help
f)   finish - apply any changes to the OAuth Service
a)   show advanced properties of the OAuth Service
d)   display the equivalent dsconfig arguments to apply pending changes
b)   back
q)   quit

Enter option [b]:
```

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

About The Policy Service

Identity Broker policies are managed by the Policy Service. The default conditions of the Policy Service can be viewed and changed with the `dsconfig` tool. For example:

- The **broker-store** option enables choosing a new location for the Broker Store.
- The **combining-algorithm** determines how decisions are made if multiple policies are applied to a request for resources. The default for the Policy Service is `deny-overrides`, which specifies that a "deny" decision from a policy should take priority over a "permit" decision. The Identity Broker also supports `permit-overrides`, `deny-unless-permit`, and `permit-unless-deny`. See the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0. August 2010* (http://docs.oasis-open.org) for details about each combining algorithm.
- The **consent-validity-duration** determines how long a consent to access data is valid once sent. Applications can specify a different validity duration for consents, which will overwrite this property.

See the *UnboundID Identity Broker Administration Guide* for details about policies and how their configuration determines data access.

To Configure the Policy Service

1. Run the `dsconfig` tool. See To Run the dsconfig Tool.
2. Select the **Policy Service** option from the UnboundID Identity Broker configuration console main menu. The following is displayed.

```
>>>> Policy Service management menu

What would you like to do?

1)  View and edit the Policy Service

b)  back
q)  quit
```

3. Choose option 1. The settings for the Policy Service are displayed.

```
>>>> Configure the properties of the Policy Service

Property                        Value(s)
-----------------------------------------
1)  broker-store                Default
2)  combining-algorithm         deny-overrides
3)  broker-store-poll-frequency 2 s
3)  consent-validity-duration   52 w 1 d

?)  help
f)  finish - apply any changes to the Policy Service
a)  show advanced properties of the Policy Service
d)  display the equivalent dsconfig arguments to apply pending changes
b)  back
q)  quit
```

4. Enter an option to change.

About Cross-Origin Resource Sharing Support

Cross-Origin Resource Sharing (CORS) enables client applications to make JavaScript requests to the Identity Broker Server (or Identity Data Store) by specifying the domain from which the request is made.These cross-domain requests are generally not allowed by web browsers without CORS support. CORS defines a way in which the browser and the server can interact to determine whether a request is coming from a trusted domain.

CORS Implementation

CORS is implemented per HTTP servlet extension. Access is governed by HTTP Servlet Cross Origin Policies defined through the `dsconfig` tool. Trusted domains can be added to these policies or defined with registered applications in the Identity Broker Console or through the `broker-admin` tool.

Note

By default, HTTP servlet extensions do not have CORS defined. Without a CORS policy defined, the configuration of the browser will determine application access.

The following are configuration options in `dsconfig`:

```
>>>> HTTP Servlet Cross Origin Policy management menu

What would you like to do?

1)  List existing HTTP Servlet Cross Origin Policies
2)  Create a new HTTP Servlet Cross Origin Policy
3)  View and edit an existing HTTP Servlet Cross Origin Policy
4)  Delete an existing HTTP Servlet Cross Origin Policy

b)  back
q)  quit

Enter option [b]:
```

HTTP Servlet Services

Enabling CORS for a particular servlet can impact another service provided by the same servlet. It is important to know which services will be affected when enabling CORS for an Identity Broker servlet. The following are available servlets and their functions.

Servlet	Functions
Identity Broker REST API Servlet	Administration of Broker Store objects, such as applications, scopes, and resources.
OAuth Servlet	OAuth authorization, token, revocation, and validation endpoints.
Policy Decision Point Servlet	XACML PDP endpoint.
Privacy Servlet	Consent management and consent history APIs.
SCIM	Profile access by data view using SCIM.

Servlet	Functions
Spring Security	Authentication and authorization layer for the rest of the servlets. Identity Broker login and registration endpoints.
UserInfo Servlet	Profile access using OpenID Connect.
Velocity	Velocity templates, including the Identity Broker's login, registration, and consent interfaces.

Note

Any servlet accepting JavaScript calls from client applications, such as the Velocity servlet, must have CORS enabled.

HTTP Servlet Cross Origin Policies

Two sample policies are available after installation. They can be associated with a servlet extension, or used as templates for additional policies.

Per-Application Origins – This policy trusts origins that are listed as trusted by applications registered with the Identity Broker.

Restrictive – This policy rejects all cross-origin requests unless explicitly defined with the `cors-allowed-origins` property. Requests from application origins that are not specified are rejected with a `403 Forbidden` return code.

Each policy accepts values for the following properties.

Property	Description
`cors-enabled`	Specifies if the CORS protocol is allowed by the servlet. The default value is `false`.
`cors-allowed-methods`	Specifies the list of HTTP methods allowed for access to resources. The default value is `GET`.
`cors-enable-per-application-origins`	Specifies that a per-application list of allowed origins (stored in the Broker Store) is consulted. The default value is `false` in the Restrictive policy and `true` in the Per-Application Origins policy.
`cors-allowed-origins`	Specifies a global list of allowed origins. If the `cors-enable-per-application-origins` property is set to `true`, and there are origins listed here, this list is consulted in addition to the per-application list. A value of "`*`" specifies that all origins are allowed. The default is an empty list.
`cors-exposed-headers`	Specifies a list of HTTP headers that browsers are allowed to access. Simple response headers, as defined in the Cross-Origin Resource Sharing Specification, are allowed. The default is an empty list.
`cors-allowed-headers`	Specifies the list of header field names that are supported for a resource and can be specified in a cross-origin request. The default values are `Origin`, `Accept`, `X-Requested-With`, `Content-Type`, `Access-Control-Request-Method`, and `Access-Control-Request-Headers`.
`cors-preflight-max-age`	Specifies the maximum number of seconds that a preflight request can be cached by the client. The default value is `1800` (30 minutes).

Property	Description
`cors-allow-credentials`	Specifies whether requests that include credentials are allowed. This value should be `false` for servlets that use OAuth2 authorization. The default value is `false`.

Assigning a CORS Policy to an HTTP Servlet Extension

CORS policies are assigned to HTTP servlet extensions through `dsconfig`.

The following are configuration options for the SCIM servlet extension:

```
>>>> Configure the properties of the Data View SCIM HTTP Servlet Extension
Property                Value(s)
-----------------------------------------------------------------------

1)  description          -
2)  cross-origin-policy  No cross-origin policy is defined and no CORS headers are recogn
ized or returned.
3)  base-context-path    /scim

?)  help
f)  finish - apply any changes to the Data View SCIM HTTP Servlet Extension
a)  show advanced properties of the Data View SCIM HTTP Servlet Extension
d)  display the equivalent dsconfig command lines to either re-create this object or only
to apply pending changes
b)  back
q)  quit

Enter option [b]: 2
```

Choose the `cross-origin-policy` option. Defined policies are listed.

```
>>>> Configuring the 'cross-origin-policy' property
The cross-origin request policy to use for the HTTP Servlet Extension.

A cross-origin policy is a group of attributes defining the level of cross-origin request
supported by the HTTP Servlet Extension.

Do you want to modify the 'cross-origin-policy' property?

1)  Keep the default behavior: No cross-origin policy is defined and no CORS headers are
recognized or returned.
2)  Change it to the HTTP Servlet Cross Origin Policy: Per-Application Origins
3)  Change it to the HTTP Servlet Cross Origin Policy: Restrictive
4)  Create a new HTTP Servlet Cross Origin Policy

?)  help
q)  quit
```

Choose the CORS policy to assign to this servlet extension.

About Dashboards and Metrics

Dashboards are configured from the Metrics Engine and display data on the Metrics page of the Identity Broker Console. Configuration is required on the Metrics Engine and the Identity Broker server to surface data in the Identity Broker Console Metrics page. Data includes:

- Performance data for the Identity Broker.
- Authorizations granted and denied to client applications.
- Consents granted, denied, and abandoned by customers.
- Most requested data.
- Most requesting client applications.

See the *UnboundID Metrics Engine Administration Guide* for steps to install the Metrics Engine. See the *UnboundID Identity Broker Administration Guide* for details about the Identity Broker Console application and the Metrics page.

To Configure the Metrics Engine and Identity Broker to show Metrics Data

This procedure assumes that an UnboundID Metrics Engine is already installed. See the *UnboundID Metrics Engine Administration Guide* for details. Make sure that the following are available:

- Make sure that the Metrics Engine was configured to use HTTPS or both HTTP and HTTPS.
- Make sure the Identity Broker is installed and configured with the `create-initial-broker-config` tool, and that the Identity Broker Console web application was installed. See To Configure the Identity Broker.
- Verify access to the Identity Broker Console at `https://<host:port>/broker-console` and log in as the administrative user.
- Click the Metrics link in the Identity Broker Console. A page with empty charts will display until the Metrics Engine is configured and data is generated.

Perform the following steps to configure the Metrics Engine:

1. From the Metrics Engine, use the `monitored-servers` tool to connect the Metrics Engine to the Identity Broker. For example:

```
./UnboundID-Metrics-Engine/bin/monitored-servers -w <ME password> add-servers \
  --remoteServerHostname <Broker host name> \
  --remoteServerPort <Broker LDAP port> \
  --remoteServerBindPassword <Broker Host Password> \
  --monitoringUserBindPassword password -p <ME LDAP port>
```

2. In a browser, access the Metrics dashboard page `https://<ME-host:https-port>/view/broker-dashboard`. Charts display (after a short period of time) with no data, as the Metrics Engine has not taken samples from the Identity Broker yet.
3. From the Identity Broker server, use the `dsconfig` tool to configure the Broker-Admin-Console web application extension for the dashboard URL:

```
./dsconfig set-web-application-extension-prop \
  --extension-name Broker-Admin-Console \
  --set dashboard-url:https://[ME-host:ME-https-port]/view/broker-dashboard
```

4. To enable the page in an inline frame, set the following:

```
./dsconfig set-velocity-context-provider-prop \
  --extension-name Velocity \
  --provider-name Dashboard \
  --add "response-header:X-Frame-Options: ALLOW-FROM
         https://<broker-host>:<broker-console-port>/"
```

   The default is `X-Frame-Options: DENY`. This value must be changed for the Metrics page in the Identity Broker Dashboard to function properly.

5. For the configuration setting to take effect, disable and then re-enable the Broker Apps Connection Handler with the `dsconfig` tool:

```
./dsconfig set-connection-handler-prop \
  --handler-name "Broker Apps Connection Handler" \
  --set enabled:false
./dsconfig set-connection-handler-prop \
  --handler-name "Broker Apps Connection Handler" \
  --set enabled:true
```

6. In a browser, access the Identity Broker Console Metrics page. The dashboard will be embedded in the page.

The sample-data-loader Tool

During the setup process, the `create-initial-broker-config` tool prompts to install default policies for the Identity Broker. See About the Installation Process and Files Installed for details about these policies.

If this is not done during the configuration process, the `sample-data-loader` tool can be used to install sample data at a later time. The `sample-data-loader` tool provides an `install` subcommand to set up the sample data and a `remove` subcommand to delete the sample data if needed.

Note: The `create-initial-broker-config` session installs two internal users, `sampleuser1` and `sampleuser2`, which are used in the sample policies. The users `sampleuser1` and `sampleuser2` corresponds to "John Public" and "Mary Private," and are installed in the backend user store repository. The user `sampleuser1` has consented to the applications, `InternalAppOne` and `ExternalAppTwo`, accessing his Customer Profile and Billing History. The user `sampleuser2` has not consented to either application.

If adding the sample data after running the `create-initial-broker-config` tool, these users must be manually added to the user store prior to running `sample-data-loader`. The following example procedure shows how to do so.

To Add Sample Users and Run the sample-data-loader Tool

1. On the backend user store, add two internal entries, `sampleuser1` and `sampleuser2`, to be used with the `sample-data-loader` tool. Or, use two existing user accounts with the `sample-data-loader`. The following shows a sample LDIF file that can be created using any text editor, and added to the Data Store using the `ldapmodify` tool.

```
dn: uid=sampleuser1,ou=People,dc=example,dc=com
objectClass: top objectClass: person
objectClass: organizationalPerson
objectClass: inetorgperson
description: This is a test user to exercise sample data within the
UnboundID Identity Broker
uid: sampleuser1
cn: Sample
sn: User1
userPassword: password

dn: uid=sampleuser2,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetorgperson
description: This is a test user to exercise sample data within the
UnboundID Identity Broker
uid: sampleuser2
cn: Sample
sn: User2
userPassword: password

bin/ldapmodify -p 1389 -D "uid=admin,dc=example,dc=com" -w passord -a -f
sample-data.ldif
```

2. On the Identity Broker, run the `sample-data-loader` tool to install the sample data.

```
sample-data-loader install \
  --trustAll --authID admin --authPassword password \
  --owner1 sampleuser1 --owner2 sampleuser2 --no-prompt
```

3. If sample data is no longer needed, run the `sample-data-loader` tool to remove the sample data.

```
sample-data-loader remove \
  --trustAll --authID admin --authPassword password \
  --owner1 sampleuser1 --owner2 sampleuser2 --no-prompt
```

Sample Requests and Policy Tests

After sample data is loaded, sample client requests can be used to test the Broker configuration. The Broker Console web application contains four Policy tests, based on the data that was loaded. See the *UnboundID Identity Broker Administration Guide* for details about running Policy Tests.

Configure the Identity Broker Console on Tomcat

The Identity Broker runs it on an embedded Jetty servlet container by default. To deploy the Identity Broker Console on Apache Tomcat, use the following procedure.

If the Profile Manager and Sample Sign-In sample applications were installed with the Identity Broker, advanced configuration options are available in the README file (`<server-root>/samples`) for each application package.

Configuring the Identity Broker Console application to use Tomcat may overwrite some of the default properties defined in:

`webapp/WEB-INF/classes/application.default.properties`

Review this file before creating an `application.properties` file for the web applications. This file can also be used as a template for creating the `application.properties` file.

1. Install Tomcat and put the WAR files for the Identity Broker Console from the Identity Broker Server's `/webapps` directory in Tomcat's `/webapps` directory.
2. Optional. Modify `$CATALINA_HOME/conf/server.xml` to set the ports. By default, they are set to 8080 and 8443, which is used by the Identity Data Store.

   ```
   <Connector port="8080" protocol="HTTP/1.1"
   connectionTimeout="20000"
   redirectPort="8443" />
   ```

3. Run `broker-admin get-application-prop` on the Broker Store to find the client ID for the Identity Broker Console application.

   ```
   $ broker-admin get-application-prop \
     --id @BrokerConsole@ \
     --property clientId \
     --script-friendly | cut -f 2


   edd75465-a41a-422c-b4d5-2d69af1de50d
   ```

4. Run the following command to determine the client secret for the Identity Broker Console. The client secret must be base64 encoded in `application.properties`, and should be removed from the file system once used.

```
$ broker-admin get-application-prop \
  --id @BrokerConsole@ \
  --property clientSecret
  --script-friendly | cut -f 2 > /tmp/secret
$ base64 encode -f /tmp/secret

S1hZMFNhUndjUwo=

$ rm /tmp/secret
```

5. Before starting Tomcat, create an `application.properties` file. This is the file that applications read to determine the Identity Broker location. Use the previously recovered client ID and secret. Save the properties file in the directory `$HOME/.broker-console`. The properties file resembles the following:

```
serviceUrl=https://<hostname>:1443
trustStoreFile=/ds/<user>/tomcat/UnboundID-Broker/config/truststore
oauthAdminClientId=30c1605d-4eb3-4403-92c4-453029e96881
oauthAdminClientSecret=eUpmUzF6SGViWQ==
```

6. Start Tomcat.
7. The Identity Broker Console redirect URI will need to be updated. See Changing the Identity Broker Console Redirect URI for details.

Changing the Identity Broker Console Redirect URI

The following set of commands updates the redirect URI for the Identity Broker Console's web application extension in the server configuration, and restarts the application's connection handler. The redirect URI is used by the Identity Broker Console itself to form the OpenID Connect request to the Identity Broker server during login.

```
$ dsconfig set-web-application-extension-prop \
  --extension-name Broker-Admin-Console \
  --set redirect-url:https://<example.com>/broker-console
```

```
 $dsconfig set-connection-handler-prop \
  --handler-name "Broker Apps Connection Handler" \
  --set enabled:false
```

```
$ dsconfig set-connection-handler-prop \
  --handler-name "Broker Apps Connection Handler" \
  --set enabled:true
```

The following command adds a registered redirect URI value for the Identity Broker Console application in the Broker Store. This redirect URI value is used by the Identity Broker's authorization endpoint to validate the redirect URI value received from the Identity Broker Console when it makes an OpenID Connect request during login.

```
$ broker-admin set-application-prop \
  --id @BrokerConsole@ \
  --add "redirectUrls:https://<example.com>/broker-console”
```

Note

The Identity Broker only accepts redirect URI values that have been registered to the application making a request. Both of these redirect URI values must match exactly.

Index

T

U