UnboundID® Identity Broker

Installation Guide

Version 4.6.0

UnboundID Corp

13809 Research Blvd., Suite 500

Austin, Texas 78750

Tel: +1 512.600.7700

Email: support@unboundid.com

UnboundID

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 UnboundID solution set provides the following:

- **Secure End-to-End Customer Data Privacy Solution** – A comprehensive identity data platform with authorization and access controls to enforce privacy policies, control user consent, and manage resource flows. The system protects data in all phases of its life cycle (create, read, update, delete as well as static/unchanging and expiring).
- **Purpose-Built Identity Data 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 platform 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, OAuth2, and OpenID Connect.

About This Guide

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

- UnboundID Privacy Suite
- UnboundID Identity Broker
- UnboundID Identity Data Store
- UnboundID Identity Proxy
- UnboundID Identity Data Sync Server
- Identity Broker API

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
- OAuth2 specification
- OAuth2 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 registries 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 OAuth2 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:

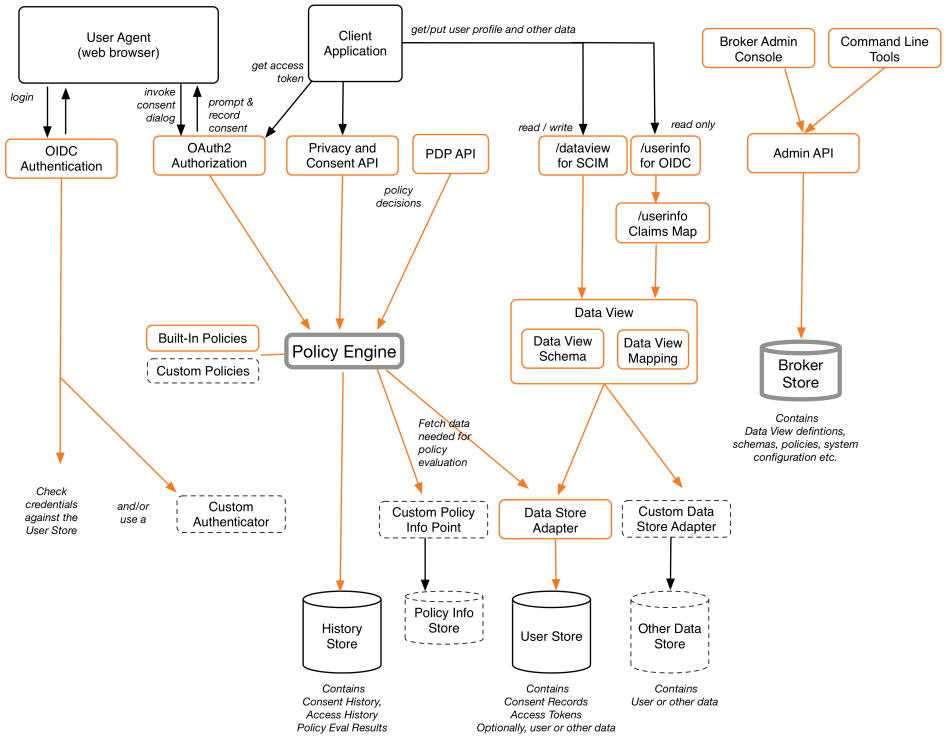
- **Support for multiple backend data stores**. The Identity Broker supports multiple data stores, with native support for the UnboundID Data Store and extension points for other data stores, such as relational databases. Applications can be written one time for access to the Identity Broker and receive 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.

- **Standards-based authorization**. The Identity Broker Server provides OAuth2-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 templates**. The Identity Broker provides a sample sign in application and other templates for implementing user authentication and consent flows. Server templates are available for further customization.

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 OAuth2.0 flow and receive access through OpenID Connect and SCIM endpoints.

The Identity Broker can also act as a relying party for identity providers such as Facebook and Google. It can either be the 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

Installation Considerations

Consider the following deployment-related issues prior to installing:

- **Determine the Identity Broker Store Topology**. The deployment determines where the Identity Broker stores its policies, Data View Schemas, and OAuth2 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.
- **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 OAuth2 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 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 6 or Java 7
- Minimum of 2 GB RAM
- UnboundID Identity Data Store 4.5 or later

There may be other required software for your system, please review the Supported Platform chart.

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 following chart lists the supported Identity Broker platforms and software versions. UnboundID does not require specific hardware.

- **Reference** – tested and confirmed that the system works as documented.
- **Yes** – the supported platform is included in UnboundID support agreements.
- **Eval Use Only** - the platform can be used to evaluate UnboundID software but should not be used for production deployments.
- **Experimental** – undergoing tests on the platform and may or may not be supported in the future.

Supported Platforms & Software

Operating Systems	Supported?	Comments
RedHat Linux 6.4	Yes	
RedHat Linux 6.5	Reference	
Solaris 10 x86 update 10	Yes	
Solaris 11.1 x86	Yes	
Solaris 11 SPARC	Yes	
AIX 7.1	Yes	
CentOS 6.4	Yes	
CentOS 6.5	Reference	
SUSE Enterprise 11 SP2	Yes	

Operating Systems	Supported?	Comments
Windows	Eval Use Only	
MacOS	Eval Use Only	

Java JDKs

JDKs	Supported?	Comments
IBM JDK 6.x 64-bit	Reference	
IBM JDK 7.x 64-bit	Yes	
Oracle JDX 6.x 64-bit	Reference	
Oracle JDX 7.x 64-bit	Reference	

Virtual Hosts/Platforms

Virtual Hosts/Platforms	Supported?	Comments
VMWare vSphere & ESX 5.1	Yes	
IBM AIX Virtualization (LPAR, PR/SM)	Yes	

App Servers/Servlet Containers

App Servers/Servlet Containers	Supported?	Comments
Apache Tomcat 7.x	Yes	
JBoss 7.x	Yes	

Identity Data Platform

Identity Data Platform	Supported?	Comments
UnboundID Identity Data Store, 4.5, or later	Yes	Required for the Broker Store
UnboundID Identity Data Proxy, 4.5	Yes	Optional
UnboundID Metrics Engine, 4.5	Yes	Optional

Browser Software

Auxiliary Software
Internet Explorer 7.0 or later
Chrome 5.0 or later
Firefox 3.0 or later
Safari 5.0 or later

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 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 generally used to store 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 can be configured to be shared by other Identity Broker instances. For each Broker Store server identified, an account for Identity Broker access is created, schema is updated to allow the storage of Identity Broker operational data, and an initial administrative account is defined for managing the Broker Store. The Identity Broker configuration will also update the schema for each LDAP user store server identified to support additional information for each user entry.

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.

Installing the Identity Data Store

The Identity Broker requires that 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.

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 labelled, `UnboundID-DS-<version>.zip`, where <version> is the latest build.
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. Select how to enable access through HTTP. This procedure assumes option 3 is chosen.

   ```
   Would you like to enable access through HTTP?

    1)  Do not configure HTTP access at this time
    2)  HTTP
    3)  HTTP with SSL
    4)  Both HTTP and HTTP with SSL

   Enter choice [1]:3
   ```

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 Key Store (JKS)
    3)  Use an existing certificate located on a PKCS12 key store
    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=co-ompany,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, data needed for OpenID Connect support, and required XACML policies. 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, which will reside on an Identity Data Store.

Three policies are disabled unless specifically enabled during the configuration process. The enabled policies are required for Identity Broker functions and should not be disabled.

- **ConsentPolicy.xml** (disabled) – Returns a decision of Permit if the resource owner has consented to allow access to all of the resources in a request.
- **GovernanceTagPolicy.xml** (disabled) – Returns a decision of Permit if the requesting application holds all governance tags held by all requested resources.
- **TrustLevelPolicy.xml** (disabled) – Returns 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.
- **AdminAccess.xml** (enabled) – Governs access to the Admin API. It ensures that only authorized applications are allowed to perform administrative actions within the Identity Broker. By default the set of authorized applications are the Broker Console, the Broker CLI, and the Privacy Preferences application.
- **DataViewFullAccess.xml** (enabled) – Determines what applications are allowed to use the "super-user" privilege from the SCIM endpoint. Super-user in this case means that requests can bypass normal policy checking. These applications are UnboundID-provided applications.

If sample data is installed, the following are performed:

- For each specified user store, two users are created over LDAP (uid=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. One instance of the Identity Broker Console application is requried. The Privacy Preferences application is optional.

- 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 key-store 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 requests. Press **Enter** to accept the default.
11. Choose the web applications to install with this instance of the Identity Broker. If this is the only instance of the Identity Broker, the Identity Broker Console must be installed. If multiple instances of the Identity Broker are installed, at least one See Web References Interfaces for a description of the Privacy Preferences / Customer Support Portal application.

    ```
    1)  Identity Broker Console
    2)  Privacy Preferences / Customer Support Portal
    3)  All of the applications
    4)  None of the applications
    ```

```
    b)  back
    q)  quit
```

12. Enter an HTTPS port to be used for the Identity Broker Console and web applications, 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
Secure Access:                 Enable SSL on LDAP Port 1443
                               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. The OAuth service's active-encryption-key value is stored in this file and should be changed before exposing the Identity Broker Server to external client applications. See About the OAuth Service for configuration information.

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-n=config`). An external store can hold user store data and/or be the location of the Broker Store.

```
Specify the credentials that the Identity Broker will
use when communicating with Broker Store and LDAP user
store instances. This tool assumes that the
credentials will be the same across all external store instances,
though you can adjust this later for each individual server using
the dsconfig tool. This entry will be created on each external
store instance when the servers are prepared in a later step.

    1)  Use cn=Broker User,cn=Root DNs,cn=config
    2)  Use a different account

    b)  back
    q)  quit
```

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.

```
Specify the base DN where the policy data should be stored

    1)  Use ou=Identity Broker,dc=example,dc=com
    2)  Use a different 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.

```
An account entry will be created under ou=Admins,ou=Identity Broker,
dc=example,dc=com for managing the broker store by users of tools
such as the Identity Broker Console and broker-admin tool.  Enter the
name (uid) of the entry to be created [admin]:
```

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

    b)  back
    q)  quit

Enter choice [3]:
```

11. A certificate is presented. Review the certificate and enter **y** to accept it.
12. 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]:
```

13. 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.

```
Enter the DN of an account on localhost:636 with which to create or
 manage the 'cn=Broker User,cn=Root DNs,cn=config' account and
configuration [cn=Directory Manager]:

Enter the password for 'cn=Directory Manager':

Created 'cn=Broker User,cn=Root DNs,cn=config'

Testing 'cn=Broker User,cn=Root DNs,cn=config' access ..... Done
```

```
Testing 'cn=Broker User,cn=Root DNs,cn=config' privileges ..... Done
Checking Broker Schema ..... Done
Initializing Broker Store ..... Done
Importing Broker Store Structure ..... Done
Verifying backend 'ou=Identity Broker,dc=company,dc=com' ..... Done
Enabling Short Unique ID Virtual Attribute ..... Done
Creating Broker Store Admin ..... Done
```

14. 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.
15. 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.
16. Enter the `host:port` for the first Identity Data Store or Identity Proxy Server.
17. Enter the `host:port` for LDAP communication with this server. The connection is validated.
18. Select an option to prepare the user store for access by the Identity Broker and press **Enter**.
19. If there are additional user data store locations, enter their `host:port`. If there are no additional servers to add, press **Enter** to continue.
20. Enter the `host:port` for LDAP communication for the additional server, or press **Enter** to continue.
21. Specify the base DN for locating user entries, such as `ou=people,dc=example,dc=com` and press **Enter**.
22. Create an LDAP search filter for this DN and press **Enter**.
23. The filter is validated against the DN. Press **Enter** (yes) to use these settings.
24. 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
  Privacy Service URL:         https://<hostname>:1443/privacy/v1

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

  Identity Broker Console:     https://<hostname>:1445/broker-console
  Privacy Preferences:         https://<hostname>:1445/privacy-preferences

  Identity Broker Location: austin

  Broker Store
    Base DN: ou=Identity Broker,dc=example,dc=com
    Broker User DN: cn=Broker User,cn=Root DNs,cn=config
    Connection Security: SSL
    Servers: <hostname>:636

  User Store
    Base DN: ou=people,dc=example,dc=com
    Search Filter: (objectClass=inetOrgPerson)
    Broker User DN: cn=Broker User,cn=Root DNs,cn=config
    Connection Security: SSL
    Servers: <hostname>:636

What would you like to do?

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

25. Press **Enter** (w) to confirm that the configuration should be applied to this Identity Broker and written to the `broker-cfg.dsconfig` file.
26. Press **Enter** to confirm the configuration.
27. 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).

```
Do you want to enable the default policies?

    1)  Yes
    2)  No
```

28. Select the option (1) 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.
29. 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.

The OAuth service's active-encryption-key value should be changed before exposing the Identity Broker Server to external client applications. This is because the initial encryption key value will be found in the `broker-cfg.dsconfig` file generated by `create-initial-broker-config`.

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. Making a configuration change with `dsconfig` or through the Identity Broker Console will prompt as to whether the change should apply to the local server only or all related servers. 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 applications to install on this Identity Broker clone. The Identity Broker Console is required at a minimum.
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. Select the server certificate option for this instance and press **Enter**.
17. To specify particular addresses on which the server will listen to client connections enter **yes**.
18. Enter **yes** to tune the JVM memory for performance. If yes, enter the amount of memory to allocate to the JVM.
19. Enter **yes** if you want to start the server after the server has been configured.
20. Review the information for the configuration, and press **Enter** to set up the server with these parameters.
21. To write this configuration to a file, press **Enter** to accept the default (yes).
22. The clone is installed and configured based on the configuration settings of the peer.

Planning a Scripted Install

The `setup` and `create-initial-broker-config` tools provide an interactive installation of the Identity Broker. If an interactive installation cannot be performed, a scripted installation can be done. To simplify the process, the `setup` and `create-initial-broker-config` tools can be run and the configuration written to the `broker-cfg.dsconfig` batch file. The batch file can then be used for scripted installs.

A successful scripted Identity Broker installation relies on the following:

- Credentials for the Broker CLI client must be generated and set in the `broker-cfg.dsconfig` batch file. Configuring the Identity Broker non-interactively requires initial configuration of the Broker Store with the `broker-admin` tool. The `broker-admin` tool requires the generated client credentials for the built-in Broker CLI application. The broker-admin tool needs to have these credentials in the server configuration as well, under the OAuth Service's `oauth-admin-client-id` and `oauth-admin-client-secret` properties.
- The default OpenID Connect scopes must be loaded. These are defined in `<server-root>/resource/install_oidc_objects.broker-admin`. The following line must be modified appropriately:

```
create-dataview-schema --set "name:Default User Schema" --set
"description:Default User Schema for OpenID Connect and SCIM" --
setFromFile "schemaJson:$SERVER_ROOT/resource/defaultUserSchema.json"
```

- The generated client credentials for the Identity Broker Console and Privacy Preferences web applications should also be written to the server configuration.

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. This creates an admin account and client credentials for built-in applications.
4. The client ID and secrets for the Broker Console, Privacy Preferences 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]
privacy-prefs-client-secret=[Privacy Preferences app client secret
```

5. Copy the command line tools, Console application, and Privacy Preferences application credentials from the `create-initial-broker-config.props` file and place them into an existing `broker-cfg.dsconfig` batch file or create the file. The following is an example of these entries:

```
dsconfig set-oauth-service-prop \
  --set active-encryption-key:[Encryption Key] \
  --set oauth-admin-client-id:[command-line tools client ID] \
  --set id-token-issuer-name:broker.example.com \
  --set oauth-admin-client-secret:[command-line tools client secret]
```

```
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]
```

```
dsconfig set-web-application-extension-prop \
  --extension-name Privacy-Preferences-App \
  --set oauth-admin-client-id:[Privacy Preferences app client ID] \
  --set oauth-admin-client-secret:[Privacy Preferences app secret]
```

6. Run the `dsconfig` command and list the `broker-cfg.dsconfig` as a batch file to configure the following:
   - Configure the web applications and include at least one Identity Broker Console application for the environment.
   - Create locations for this Identity Broker and any additional Identity Broker servers.
   - Create the external server client access to the Identity Data Store.
   - Create Data Views.
7. Substitute the Identity Broker server root path into the `<server-root>/resource/install_oidc_objects.broker-admin` file.
8. Load `<server-root>/resource/install_oidc_objects.broker-admin` using the `broker-admin` tool.
9. Import policies with the `broker-admin import-policy` tool. See About the Installation Process and Files Installed for a list of mandatory policies.
10. After installing the policies, run `broker-admin` in batch mode with the `resource/install-data-view-mappings.broker-admin` file as input.
11. Use the `broker-admin` tool to load any other Broker Store data, such as sample data. See About the Command Line Tools.
12. Validate that the Broker Console and Privacy Preferences applications are configured properly by logging in as the admin user.

    When finished, delete the `<server-root>/tmp/create-initial-broker-config.props` file.

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: Configuration

During the setup process, the Identity Broker's `setup` tool invokes the `create-initial-broker-config` script, configuring the communication between the Identity Broker and its repositories. Additional tools are available to manage and configure Identity Broker components.

This chapter provides additional, optional Identity Broker tools and configuration and includes the following:

Identity Broker Command-Line Tools

The command-line tools are located in the `/bin` directory and provide 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:

- **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 Data Views and 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.

The dsconfig tool

The `dsconfig` tool 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. The following can only be done with the `dsconfig` tool after an initial Identity Broker configuration:

- Adding Data Views to the Identity Broker. Data Views use SCIM schemas to enable attribute mapping from one or more Identity Data Stores to the data collected through the

Identity Broker. Once added, Data Views can be edited in the Identity Broker Console.

- Adding additional Data Stores to the Identity Broker environment.

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

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

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

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 Store Adapters

Store adapters interface with backend data stores. Store adapters have the same API as the Data View, except that store adapters have the option to support authentication and/or user metadata attributes. There must be *at least one* store adapter that supports user metadata and authentication for each Data View.

Store adapters expose data in the native SCIM objects. A JDBC store adapter might return SCIM objects where attribute names are JDBC-specific database names, such as `employee_id`, `first_name`, and `last_name`. The LDAP store adapter returns SCIM objects with LDAP-specific attribute names, such as `givenName`, `sn`, and `cn`. The Identity Broker Console is used to map these adapter SCIM objects to the Data View schema.

About User Metadata

The Identity Broker stores OAuth tokens, auth codes, and consents in an operational attribute called `userMetaData` that is added to a user's entry within a User Store. The `userMetaData` attribute is configured per store adapter and can be stored in any format. At least one store adapter must support storing the user metadata attribute in an Identity Broker environment.

Metadata is divided into small and large attributes. Small metadata houses tokens, auth codes and consents. Large metadata stores a user's consent history. This separation enables the Identity Broker to access only those elements needed. 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: JHR1c3Rpbmc3ODk=
id-broker-user-large-metadata: YXNkZ2YgYXNkIGFzZGYgYXNkZ2Fkc2ZaGFk
ZmhhZHNmaGFkc2ZoYWRzZmhhc2RmZ2FzZGYgYXNkZ2FzZGdoYXNkZmhhc2RnYXNkZ2
hhc2ZkaGFzZGZnYXNkZ2ggc0RGSEFTREZIQVNERkhzZGdzREcgQVNEIEdBU0RHIEFE
U0ZIIEFTUkhBU0RGQVNEIDIgQVNERkcgQVNEIFRRV1RHU0dBU0RH
```

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, Oracle DSEE, or Microsoft Active Directory.

The LDAP store adapter uses the SCIM SDK to provide options such as virtual list view, pagination, and functionality like Identity Proxy load-balancing algorithms. Configure an LDAP Store Adapter using the `dsconfig` tool.

The LDAP Store Adapter involves the following configuration parameters:

- **ObjectClass**. Determines the native schema to expose. For example, the `InetOrgPerson` schema provides LDAP attributes and `UserObjectClass` provides Active Directory Attributes. The schema will not include operational attributes, but they can be explicitly included using the `include-operational-attribute` setting on the LDAP Store Adapter.
- **Base DN**. Determines the scope of the data within a subtree.
- **Search Filter**. Determines if a search filter is used match specific items in the tree.
- **Create-DN Pattern**. Determines if create should be supported.
- **SCIM ID Attribute**. Determines which attribute to use as the SCIM ID.
- **Operational Attributes**. Determines if operational attributes should be included.
- **Load-Balancing Algorithm**. Specifies the load balancing algorithm.
- **User MetaData Attributes**. Determines the attributes to use for metadata. At least one store adapter MUST be able to store the user metadata attributes.

Configuring Store Adapters

The Identity Broker comes with an LDAP store adapter that can be used to interface with backend data 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. Configure Store Adapter(s) and Data View using the Identity Broker Console or the `dsconfig` command.

About the 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
```

About Data Views

Data Views provide a unified profile, enabling the Identity Broker server to present user attributes from disparate sources as a single identity. Data Views rely on a single, SCIM-based schema that can map one or more user stores to the resource defined in the Data View. For example, a Data View can be created for attribute `displayName` that maps that attribute to three existing user stores. By editing the Data View in the Identity Broker Console, this attribute can be mapped to the attributes that are surfaced for each user store.

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.

Data View – A Data View is created using the `dsconfig` tool and is associated with a Data View Schema of the same name. Once the Data View is created, it can be edited in the Identity Broker Console. Attributes from the associated Data View schema are mapped to the attributes from the associated user store or stores.

To Configure Data Views

Configuring Data Views is a multi-step process. This step in the process relies on the existence of a Data View Schema in the Broker Store, on which this new Data View will rely. Data View Schemas are SCIM schemas created in JSON format. They are imported into the Broker Store from the Identity Broker Console. See the *UnboundID Identity Broker Administration Guide* for details.

1. Start the `dsconfig` tool with the following command:

   ```
   $ bin/dsconfig
   ```

2. Enter the required connection information to the Identity Broker server. See To Run the dsconfig Tool for details.

3. When the Identity Broker configuration main menu displays, type the Data View option (3) and press **Enter**.

4. Select an option from the Data View management menu.

   ```
   >>>> Data View management menu
   What would you like to do?

   1)  List existing Data Views
   2)  Create a new Data View
   3)  View and edit an existing Data View
   4)  Delete an existing Data View

   b)  back
   q)  quit

   Enter option [b]:
   ```

5. Choose Create a new Data View (2), and press **Enter**.

6. If user stores were configured with the `create-initial-broker-config` tool, a default Data View was created with that store. Press **Enter** to choose the default (n) use an existing Data View as a template.

7. Choose the Data View to use as a template and press **Enter**.

8. Specify a name for the Data View Schema that will be associated with the new Data view and press **Enter**. The name must exactly match the "name" attribute for a Data View Schema that exists in the Broker Store. Typically the name describes the resource type, such as "User" or "Subscriber." The tool will later verify that this schema is present in the Broker Store.

9. Configure the properties of the Data View.

   ```
   >>>> Configure the properties of the Data View
   >>>> via creating '<new-name>' Data View

   Property              Value(s)
   ----------------------------------------
   1)  description           -
   ```

```
2)  enabled               true
3)  dataview-schema-name  <new-name>
4)  store-adapter         UserStoreAdapter

?)  help
f)  finish - create the new Data View
a)  show advanced properties of the Data View
d)  display the equivalent dsconfig arguments to create this object
b)  back
q)  quit

Enter option [b]:
```

10. Define or adjust any of the properties. When finished, select the create new Data View option and press **Enter**.
11. If the Identity Broker was defined to keep its configuration synchronized with other servers, a prompt displays to update the current server or all servers. Choose an update option and press **Enter**.
12. Open the Identity Broker Console **Data Classification** section to map attributes from the Data View Schema to the related user stores in the new Data View.

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 Data View and correlation attribute is used to map the store adapters for the user store and the metadata store. 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.

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)   active-encryption-key                ********
2)   alternate-decryption-key             The active-encryption-key will be the only key
used for decryption
3)   authorization-code-validity-duration  1 m
4)   access-token-validity-duration        12 h
5)   refresh-token-validity-duration       4 w 2 d
6)   reuse-refresh-tokens                  true
7)   user-approval-page-url                /view/oauth/approve
8)   error-page-url                       /view/oauth/error
9)   id-token-validity-duration            15 m
10)  id-token-issuer-name                 vm-medium-73.unboundid.lab
11)  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]:
```

Notes:

- The encryption and decryption keys are used to protect Broker Store tokens so that values cannot be determined if the Broker Store is compromised. All Identity Broker

Servers that share a Broker Store must use the same encryption key.

- If an encryption key changes, use the `alternate-decryption-key` setting to specify the previous key. This ensures that any client applications using the authorization code and token values encrypted with the previous key can still be decrypted.

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.

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)  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.
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

Property	Description
	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).
`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    /dataview

?)  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. 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
```

5. 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.

Customizing the Identity Broker Web Applications

The Identity Broker can be installed with the Identity Broker Console and Privacy Preferences web application for end-users to view the set of applications and resources to which they have given consent with the added option to revoke that consent if required. Depending on the user's privileges, the Privacy Preferences application can also be used by customer support staff to assist end-users with their consent management. By default, applications are deployed through an embedded Jetty servlet container.

Note

If bookmarking the application pages, bookmark the Identity Broker Console and Privacy Preferences landing pages, not the login page. Bookmarking the login page causes authentication errors.

Customizing the Identity Broker Console Login Pages

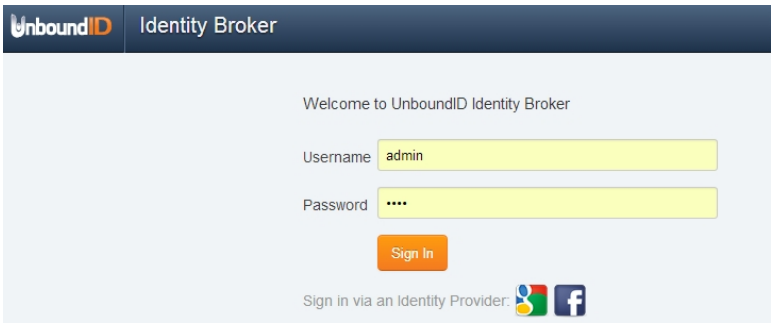
The Identity Broker supports HTTP-accessible web page hosting using the Velocity Template Language (VLT). Velocity is an open-source project of the Apache Software Foundation, which uses VTL code statements to reference dynamic content within a web page. See Velocity Templates for details.

The Identity Broker web pages can be customized to serve template-generated content, static content (images, CSS, and Javascript files), or runtime information about the server, its data, schema, or any other information. Velocity templates are obtained from the filesystem through a loader instance which is selected for each request based on the request's Accept header, and whether the loader has access to a resource that can fulfill the requested page. For more information about Velocity and the Velocity Template Language, see `http://velocity.apache.org`.

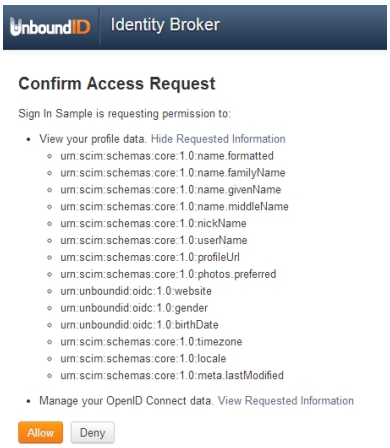
The Identity Broker hosts three pages through which end users can log into the system, manage consent, and see errors. These pages can be rebranded to better represent a company or department.

The pages are implemented as Velocity templates in the Identity Broker server's directory `<server-root>/config/velocity/templates`, whose variable values are provided by the server when the page is accessed. The three templates are:

- **login.vm**. The OAuth login page that end users are directed to if they are not currently logged in. This is the page that a client application will serve to enable login.



- **oauth/approve.vm**. The page where end user's can see information about a request to access their resources and either allow or deny access. This page can be customized for client applications.



- **oauth/error.vm**. An error page is available that should rarely be seen by end-users.

The directory `<server-root>/config/velocity/statics` can also be used by the pages to store images, CSS files or anything else that is not within the Velocity Context.

The server exposes the various objects to the templates that contain useful information at request time. The templates can access bean properties and methods of each of these. For example, the template can access the application's name using the variable `$application.name`. The following objects are exposed by the server:

- **principal** – Information about the currently logged in user. Will be null the user is not currently logged in.
- **authorizationRequest** – The authorization request being made by an application.
- **application** – The application making the request for resources.
- **scopes** – One or more scopes being requested. Each scope defines a set of resources, purpose, and actions.

- **isOfflineAccess** – Boolean where true indicates that the application is requesting permission to the approved scopes when the user is not online.
- **isForceConsent** – Boolean where true indicates the resource server will prompt the end-user for permission each time information is returned to the application making the request.

In addition, the following Velocity Tools are available for general use:

- display
- escape
- convert

See Velocity Tools Context Provider for more information.

Customizing a Web Application Logo

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

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

```
$HOME/.privacy-preferences/branding-override.css
```

If these files are present, the Identity Broker uses these to overwrite existing style sheets.

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

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

Style changes take affect after an application is restarted.

Running the Broker Applications on Tomcat

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

To Configure the Identity Broker Web Applications on Tomcat

Configuring the Identity Broker web applications to use Tomcat may overwrite some of the default properties as 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 and Privacy Preferences apps 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. The same command can be used for the Privacy Preferences 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. Run the same command for the Privacy Preferences application. 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 @PrivacyPrefs@ --property clientSecret --sc
ript-friendly | cut -f 2 > /tmp/secret
$ base64 encode -f /tmp/secret
```

```
S1hZMFNhUndjUwo=

$ rm /tmp/secret
```

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

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

7. Repeat the previous step for the Privacy Preferences application and save the file to the directory `$HOME/.privacy-preferences`. The properties file resembles the following for the Privacy Preferences application:

```
serviceUrl=https://<hostname>:1443
trustStoreFile=/Users/<user>/test/broker/UnboundID-Broker/config/keystore
scimDisplayNamePath=urn:scim:schemas:core:1.0:name.formatted
scimResourceName=user
scimUserNamePath=urn:scim:schemas:core:1.0:userName
scimQueryContainsEnabled=true
oauthAdminClientId=bb1f8875-9c6c-44da-b033-0d324727ab13
oauthAdminClientSecret=V0lXYnFOd2wzVQ==
```

8. Start Tomcat, and go to the Broker Console's URL, `http://<localhost>:8080/broker-admin-console`
9. Do the same thing for the Privacy Preferences app: `http://<localhost>:8080/privacy-preferences`

Velocity Templates

The Metrics Engine exposes Velocity pages through an HTTP Servlet Extension. If the HTTP Connection Handler is enabled, the Velocity extension is enabled.

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

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

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

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

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

- **description** – A description of the extension.
- **cross-origin-policy** – Defines a cross origin policy for this extension.
- **base-context-path** – URL base context for the Velocity Servlet.
- **static-content-directory** – In addition to templates, the Velocity Servlet will serve miscellaneous static content related to the templates. By default this is `config/velocity/statics`.
- **static-custom-directory** – If static content is customized, it resides in `velocity/statics` by default.
- **template-directory** – The template directory from which templates are read. By default this is `config/velocity/templates`. This directory also serves as a default for Template Loaders that do not have a template directory specified.
- **static-context-path** – URL path beneath the base context where static content can be accessed.
- **allow-context-override** – Indicates whether context providers may override existing context objects with new values.
- **mime-types-file** – Specifies a file that is used to map file extensions of static content to a Content Type to be returned with requests.
- **default-mime-type** – The default Content Type for HTTP responses. Additional content types are supported by defining on or more additional Velocity Template Loaders.

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

```
http://example.com:8080/view/hello?name=Joe
```

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

```
<html>
  <body>
    Hello $ubid_request.parameters.name
  </body>
</html>
```

A pop-up window displays a table on the page that lists all variables that are in the Velocity Context. References like `$ubid_request` can appear in the template file and be replaced when the template is rendered. This information can be used to check which variables are permitted to be in the template along with the variable values.

A debug option can be used in any Velocity template for verifying available information in the Velocity Context:

```
parse("_debug.vm")
debug()
```

If a variable is added to a template for something that does not exist, the rendered page will contain a literal string of the unfulfilled variable (for example `$undefined_variable`).

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

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

Supporting Multiple Content Types

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

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

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

The ability to serve multiple formats of a document to clients at the same URL is typically called *content negotiation*. HTTP clients indicate the type of content desired using the `Accept`

header. A client may use a header like the following to indicate that they prefer content in XML but will fallback to HTML if necessary:

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

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

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

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

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

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

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

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

Velocity Context Providers

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

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

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

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

Velocity Tools Context Provider

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

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

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

Chapter 5: 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:

Running the Identity Broker

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

Stopping 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
```

To Schedule a Server Shutdown

The Identity Broker provides the capability to schedule a shutdown and send a notification to the `server.out` log file. The following example sets up a shutdown task that is schedule to be processed on April 3rd, 2013 at 11:00pm CDT. The server uses the UTC time format if the provided timestamp includes a trailing "Z", for example, 201304032300Z. The example also uses 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 201304032300Z
```

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 provides an interactive method to remove the components from the system.

To Uninstall the Identity Broker

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

   ```
   $ ./uninstall
   ```

2. 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
   ```

3. 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]:
   ```

4. Manually delete any remaining files or directories.

Updating the Identity Broker and the Broker Store

When updating the Identity Broker product, the Broker Store data is expected to be at least as current as the Identity Broker version. Upgrading the Broker Store may add new schema

elements and default configuration elements essential in keeping the Broker service running. Since the Identity Broker will not start if the Broker Store is a version older than the Identity Broker itself, update the Broker Store then update instances of the Identity Broker.

Note

Upgrade is supported for post 4.6.0 releases.

Perform the following steps to update the Broker Store:

1. Before updating the Broker Store, use the `backup` tool to backup the Broker Store data.
2. Gather information about the 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, run the command on one of the servers:

```
prepare-external-store --update --isBrokerStore --brokerStoreBaseD
   <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. Use the `update` tool to update the local Identity Broker instance.
7. Test the new Identity Broker installation. There may be other manual, post-update steps necessary such as re-importing policy files. See the Identity Broker release notes for information specific to this release.
8. Update any other Identity Broker instances.

Chapter 6: Reference

This chapter provides general reference about various files and components of the Identity Broker.

This section includes:

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.
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.

Directories/Files/Tools	Description
webapps	Stores the war files for reference implementations (privacy preferences and the admin console)

The 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.
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.
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.
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.

Tool	Description
oauth2-request	Performs OAuth2.0 requests on the Identity Broker. This tool can be used to test OAuth2.0 functions of the Identity Broker, and to manage OAuth2.0 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 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.
summarize-config	Generate a configuration summary of either a remote or local Identity Broker instance. By default, only basic components and properties will be included. To include advanced component, use the `--advanced` argument.

Index