OpenDS

Configuration Guide

Version 0.1
July 2006

Contents

Overview

The OpenDS Directory Server is a pure Java implementation of a network-accessible database that is able to store information in a hierarchical form. Clients may communicate with it using a standard protocol (e.g., LDAP) to retrieve and update information in a variety of ways. We aim to provide unmatched performance and scalability, robust error handling and debugging capabilities, easy extensibility, and innovative feature sets.

This document provides information about the underlying configuration for the OpenDS Directory Server. It includes both the structure of the configuration and the individual settings that may be provided.

Copyright and trademark information

Feedback

Please direct any comments or suggestions about this document to:
dev@OpenDS.dev.java.net

Acknowledgements

The general format of this document was based on the documentation template used by OpenOffice.org.

Modifications and Updates

Date	*Description of Change*
July 2006	Initial draft

The OpenDS Configuration Hierarchy

Like the Sun Java System Directory Server, the OpenDS Directory Server treats the configuration in a manner similar to user data. In particular, the data is managed in a directory hierarchy (with a base DN of "cn=config"), and the configuration is accessible via directory protocols like LDAP. The base "cn=config" entry contains much of the core server configuration, and the configuration for the remaining components is organized below that base entry. The general structure of the OpenDS Directory Server configuration is as follows:

- cn=config
  - cn=Access Control Handler
  - cn=Account Status Notification Handlers
  - cn=Alert Handlers
  - cn=Backends
  - cn=Connection Handlers
  - cn=Entry Cache
  - cn=Extended Operations
  - cn=Identity Mappers
  - cn=Loggers
  - cn=Matching Rules
  - cn=Monitor Providers
  - cn=Password Generators
  - cn=Password Policies
  - cn=Password Storage Schemes
  - cn=Password Validators
  - cn=Plugins
  - cn=Root DNs
  - cn=Root DSE
  - cn=SASL Mechanisms
  - cn=SSL
  - cn=Synchronization Providers
  - cn=Syntaxes
  - cn=Virtual Attributes
  - cn=Work Queue

Each of these portions of the configuration will be addressed in subsequent sections of this document.

Accessing the OpenDS Configuration

In the future, the process of managing the OpenDS Directory Server will be performed through user-friendly tools that provide a stable and supported interface for configuration changes, administrative operations, and monitoring. These interfaces will likely include:

- A set of command-line tools that can be used as a lightweight administration interface and accessed in user-defined scripts for performing common operations.
- One or more graphical interfaces. This will likely include a Web-based administration framework, and may also include a "fat" client that does not require browser-based access.
- An administration shell, which will provide a text-based interface to the configuration in a manner that is more interactive than the command-line tools.
- One or more APIs that will allow for a programmatic interface for administrative capabilities.

At the present time, however, none of these administrative interfaces have been developed and therefore all access to the configuration must be performed either by editing the configuration file with the server offline, or by interacting with the server over a protocol like LDAP or JMX while it is online. Note that these methods expose the underlying implementation detail and are subject to change without warning in the future. The official administrative interfaces that we provide in the future should provide a greater degree of stability so that the impact of changes in the underlying implementation can be minimized between releases.

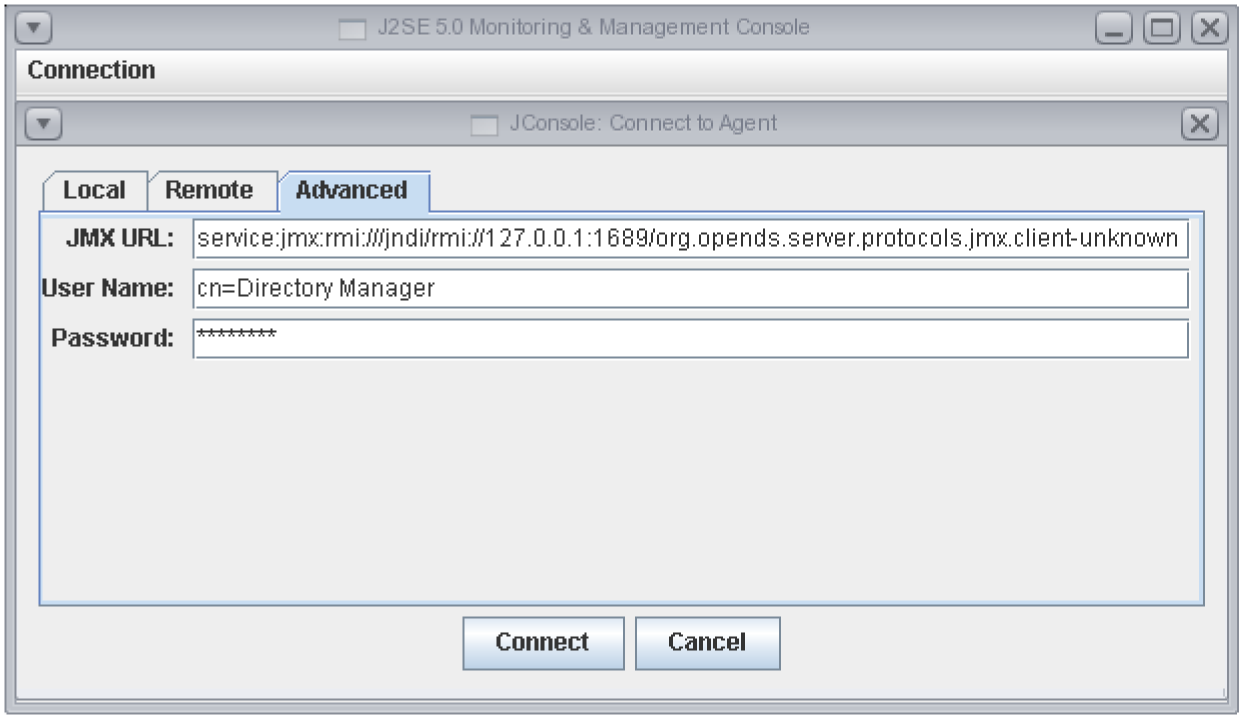
The OpenDS Directory Server has been designed so that it has an extensible mechanism for interacting with the configuration. The default implementation stores the configuration in an LDIF file (much like the Sun Java System Directory Server), but it is possible to replace that with an alternate repository (e.g., an embedded database or a centralized remote data store). Using the default LDIF-based data store, the configuration can be edited while the server is offline. It should not be edited while the server is online, as those changes may be lost if other updates are made through another means.

A safer option for making configuration changes is through the modification of the associated entries over a protocol like LDAP or JMX. Configuration changes can be made over LDAP using any standard client like ldapmodify. Unlike the Sun Java System Directory Server, none of the entries below the configuration use the extensibleObject objectClass, and therefore all changes will be subject to schema validation (which protects against the use of undefined attributes or placing configuration attributes in the wrong entries). The OpenDS Directory Server also does a better job of validating configuration changes, so that invalid values are more likely to be rejected.

The OpenDS Directory Server also provides access to configuration information using JMX (the Java Management Extensions). The Java Development Kit provides the jconsole utility, which acts as a JMX client and can be used to access the OpenDS Directory Server configuration and monitor information. Because OpenDS uses advanced JMX functionality, it is necessary to provide jconsole with a special URL to use to interact with the server. That URL is:

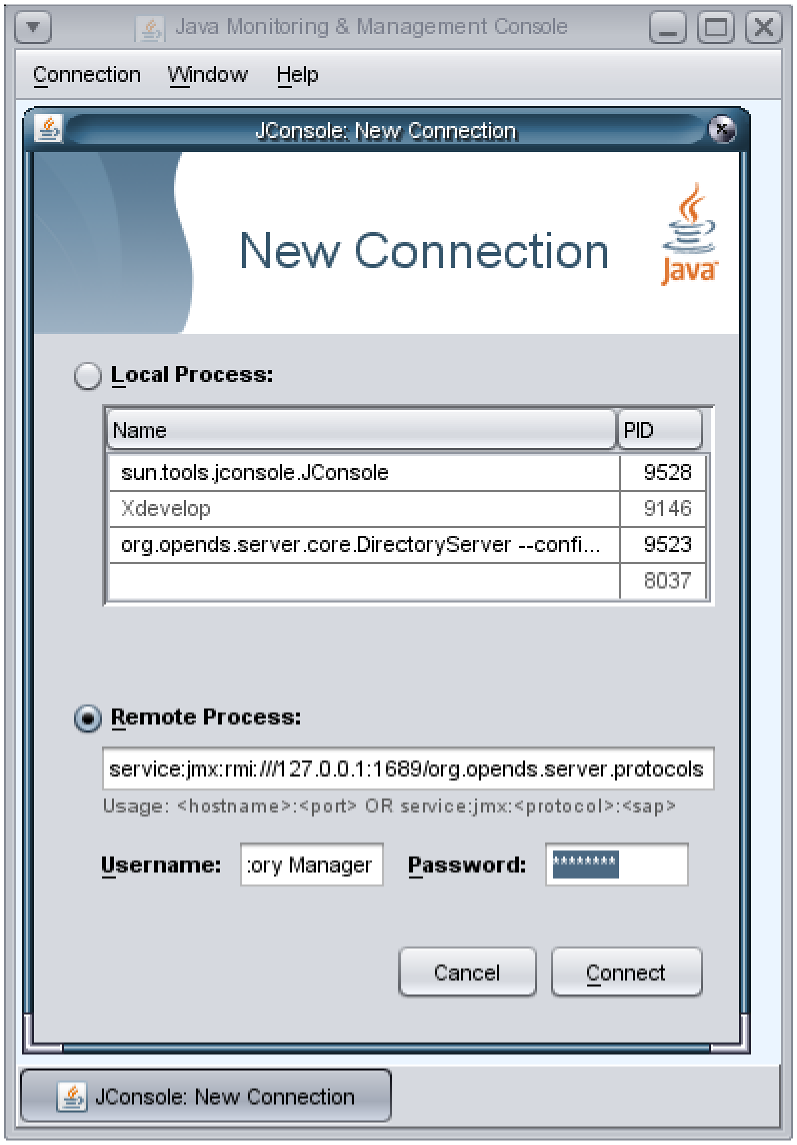
service:jmx:rmi:///jndi/rmi://*host*:*port*/org.opends.server.protocols.jmx.client-unknown

where *host* is the address (IP address or resolvable hostname) of the system running the OpenDS Directory Server instance, and *port* is the port on which the JMX connection handler is listening for connections (the default port is 1689). When using the version of jconsole provided with Java 5, this URL should be entered in the "JMX URL" field of the "Advanced" tab:


J2SE 5.0 Monitoring & Management Console
Connection
JConsole: Connect to Agent
Local
Remote
Advanced
Connect
Cancel

JMX URL: service:jmx:rmi:///jndi/rmi://127.0.0.1:1689/org.opends.server.protocols.jmx.client-unknown
User Name: cn=Directory Manager
Password: *********

For the version of jconsole provided with Java 6, enter the URL in the text field immediately below the "Remote Process" label:



Local Process:

Name	PID
sun.tools.jconsole.JConsole	9528
Xdevelop	9146
org.opends.server.core.DirectoryServer --confi...	9523
	8037

[✓ Remote Process:]

service:jmx:rmi:///127.0.0.1:1689/org.opends.server.protocols

Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap>

Username: :ory Manager Password: ********

Note that a bug in the Java 6 version of jconsole (CR 6387250, which applies to Java 6 builds 86 through 88, including Java 6 Beta 2) may prevent the use of that version of the jconsole tool from being used to access the OpenDS configuration and monitoring data. If you will be using Java 6, then it is generally recommended that it be the most recent build.

In either case, it will also be necessary to provide the username and password for a user that is authorized to access the server over JMX. By default, all users are allowed to access the server using JMX, so the username field may include the DN of any user defined in the server, and the password field should have the password for that user.

The Core Configuration

The "cn=config" entry in the OpenDS Directory Server configuration is used to hold general parameters that are applicable to the entire server, regardless of the location of the data being accessed and the mechanism used to interact with it. Attributes contained in this entry include:

- `ds-cfg-add-missing-rdn-attributes` -- This indicates whether the server should automatically update the an entry being added to the server so that its attribute set includes all attribute value pairs included in the RDN for that entry. According to the original LDAPv3 specification (RFC 2251, section 4.7) the client must include all of these attributes in the add request, although the revised specification (RFC 4511, section 4.7) it is listed as optional.

- `ds-cfg-allow-attribute-name-exceptions` -- This indicates whether the server should allow schema elements to contain names that are technically invalid according to the LDAP specification. Although schema element names should start with an ASCII alphabetic character and should only contain ASCII letters, numeric digits, and hyphens, setting the value of this configuration attribute to "true" will also allow schema element names to begin with numeric digits and include the underscore character. This option is primarily used for backward compatibility with the Sun Java System Directory Server, and it is not recommended that this capability be enabled unless it is required for applications that expect to use invalid schema element names.

- `ds-cfg-bind-with-dn-requires-password` -- This indicates whether simple bind requests will be allowed to contain a DN if no password is given (i.e., it is a zero-length string). While this is technically a valid way to perform an anonymous bind, it can expose security problems in some applications. It is recommended that this option be left enabled unless one or more clients need to include a DN in an anonymous bind request.

- `ds-cfg-check-schema` -- This indicates whether the OpenDS Directory Server should ensure that all operations result in entries that are valid according to the defined server schema. It is strongly recommended that this option be left enabled to prevent the inadvertent addition of invalid data into the server.

- `ds-cfg-default-password-policy` -- This specifies the DN of the configuration entry for the password policy that should be used for all users in the directory that do not use an alternate password policy via the `pwdPolicySubentry` operational attribute. See the Password Policy Configuration section for more information on configuring password policies in the OpenDS Directory Server.

- `ds-cfg-invalid-attribute-syntax-behavior` -- This specifies the behavior that the OpenDS Directory Server should exhibit when it encounters an attribute value that violates the syntax of the associated attribute type. The default value for this configuration attribute is "reject", which will cause any attribute value that violates the associated syntax to be refused. Other values that may be used include "accept", which will silently allow the value, and "warn", which will accept the value but generate a warning message in the server's error log. It is recommended that the default "reject" behavior be used unless there are known cases in which applications may provide attribute values that violate the associated syntax.

- `ds-cfg-notify-abandoned-operations` -- This indicates whether a response should be sent to a client if a requested operation is abandoned. According to the LDAP specification, operations that are abandoned should not receive any notification but this may cause problems with applications that always expect to receive a response to the requests that they send. If the value of this configuration attribute is changed to "true", then operations that are successfully abandoned will receive a response with a result code of 118 (canceled). Note that this applies only to operations that are abandoned and does not have any impact on operations that are interrupted with the LDAP cancel extended operation, as those operations must always receive a response.

- `ds-cfg-proxied-authorization-identity-mapper-dn` -- This specifies the DN of the configuration entry for the identity mapper that should be used to identify the target user specified in a proxied authorization control. See the Identity Mapper Configuration section for more information on configuring identity mappers in the OpenDS Directory Server.

- `ds-cfg-single-structural-objectclass-behavior` -- This specifies the behavior that the OpenDS Directory Server should exhibit when it encounters an entry that does not have exactly one structural objectclass. According to the LDAP specification, all entries must have exactly one structural objectclass, and certain advanced functionality (e.g., name forms and DIT content rules) depend on this. However, some directory servers do not enforce this policy and may allow entries that do not contain any structural objectclasses, or entries that have multiple structural objectclasses. The default value for this configuration attribute is "reject", which will cause the server to refuse any entry that does not contain exactly one structural objectclass, and it is recommended that this value be used unless there are known cases in which applications may create invalid entries. Other values that may be used for this attribute include "accept", which will silently allow the entry, and "warn", which will accept the entry but generate a warning message in the server's error log.

- `ds-cfg-size-limit` -- This specifies the maximum number of entries that may be returned to a user for a search operation. A value of zero indicates that there is no limit on the number of entries that may be returned. Note that this server-wide setting may be overridden on a per-user basis by including the `ds-rlim-size-limit` operational attribute in the user's entry with an alternate value.

- `ds-cfg-time-limit` -- This specifies the maximum length of time in seconds that the server should spend while processing a search operation. A value of zero indicates that there is no limit on the length of time that the server may spend. Note that this server-wide setting may be overridden on a per-user basis by including the `ds-rlim-time-limit` operational attribute in the user's entry with an alternate value.

- `ds-cfg-writability-mode` -- This specifies the behavior that the server should use when it receives a request to alter the contents of an entry (e.g., via an add, delete, modify, or modify DN operation). The default value for this configuration attribute is "enabled", which will allow the associated write operation (if it is otherwise valid and the user has permission to perform the requested operation). Other values that may be used for this attribute include "disabled", which will cause the server to reject all write operation requests including those associated with synchronization or internal operations, and "internal-only", which will cause the server to reject all write operation requests from external clients but will allow those associated with synchronization or internal operations.

The Access Control Handler Configuration

The "cn=Access Control Handler,cn=config" configuration entry defines the access control handler that will be used for the OpenDS Directory Server. The access control handler is defined through an extensible interface, so it is possible for alternate implementations to be created. However, at the present time, the access control implementation for the OpenDS Directory Server has not yet been developed and therefore all requested operations will always be allowed (including those requested by anonymous or unauthenticated users).

The Account Status Notification Handler Configuration

The "cn=Account Status Notification Handlers,cn=config" configuration entry is the parent entry for all account status notification handlers defined in the OpenDS Directory Server. Account status notification handlers may be used to provide notification to users in some form (e.g., by an e-mail message) when the status of a user's account changes in some way. Examples of account status notification types include:

- The account has been temporarily locked due to too many failed authentication attempts.
- The account has been permanently locked due to too many failed authentication attempts.
- The account has been unlocked by an administrator.
- An authentication attempt has failed because the account has been locked after remaining idle for too long.
- The account has been administratively disabled.
- The account has been administratively re-enabled.
- An authentication attempt has failed because the password for the associated account has expired.
- An authentication attempt is successful, but the user's password will expire in the near future.
- A user's password has been changed by that end user.
- A user's password has been reset by an administrator (i.e., it was changed by anyone other than that end user).

At the present time, the account status notification handler configuration is not yet complete and therefore no such notifications will be provided in any of these conditions.

The Alert Handler Configuration

The "cn=Alert Handlers,cn=config" configuration entry is the parent entry for all alert handlers defined in the OpenDS Directory Server. Alert handlers may be used to notify administrators of any significant events (e.g., the server is being started or shut down) or problems (e.g., an error occurred while attempting to write an updated configuration) that are detected by the server through some means.

All alert handler configuration entries must contain the `ds-cfg-alert-handler` structural objectclass, which requires the following configuration attributes:

- `ds-cfg-alert-handler-class` -- This specifies the fully-qualified name of the Java class that provides the alert handler functionality. This class must provide a concrete implementation of the `org.opends.server.api.AlertHandler` interface.

- `ds-cfg-alert-handler-enabled` -- This indicates whether the alert handler should be enabled. If the alert handler is not enabled, then it will not be used to generate alert notifications.

At the present time, only a single alert handler has been implemented, which will generate JMX notifications based on the alerts that it receives. There are no additional configuration attributes that may be used in conjunction with the JMX alert handler.

The Backend Configuration

The "cn=Backends,cn=config" configuration entry is the parent entry for all backends defined in the OpenDS Directory Server. Backends are responsible for providing access to the underlying data presented by the server. The data may be stored locally (e.g., in an embedded database), remotely (e.g., in an external system), or generated on the fly (e.g., calculated from other information that is available).

All backend configuration entries must contain the `ds-cfg-backend` structural objectclass, which defines the following configuration attributes:

- `ds-cfg-backend-class` -- This specifies the fully-qualified name of the Java class that provides the backend functionality. This class must be a concrete subclass of the `org.opends.server.api.Backend` superclass.

- `ds-cfg-backend-enabled` -- This indicates whether the backend should be enabled. If the backend is not enabled, then it will not be used when processing operations.

- `ds-cfg-backend-id` -- This provides a name that will be used to identify the associated backend. It must be unique among all backends in the server.

- `ds-cfg-backend-base-dn` -- This specifies the base DN(s) for the data that the backend will handle. A single backend may be responsible for one or more base DNs. Note that no two backends may have the same base DN, although one backend may have a base DN that is below a base DN provided by another backend (similar to the use of sub-suffixes in the Sun Java System Directory Server).

- `ds-cfg-backend-writability-mode` -- This specifies the behavior that the backend should use when processing write operations. A value of "enabled" will allow write operations to be performed in that backend (if the requested operation is valid, the user has permission to perform the operation, the backend supports that type of write operation, and the global `ds-cfg-writability-mode` attribute is also set to "enabled"). A value of "disabled" will cause all write attempts to fail, and a value of "internal-only" will cause external write attempts to fail but will allow writes by synchronization and internal operations.

There are several types of backends defined in the OpenDS Directory Server, and they will be described individually in the remainder of this section.

The Berkeley DB Java Edition Backend

The primary backend provided by the OpenDS Directory Server uses the Berkeley DB Java Edition to store user-provided data in a local repository. It is the traditional "directory server" backend and is similar to the backends provided by the Sun Java System Directory Server. It stores the entries in an encoded form and also provides indexes that can be used to quickly locate target entries based on different kinds of criteria. The `org.opends.server.backends.jeb.BackendImpl` class provides the implementation for this backend.

Because of the number of configuration options that are available for this backend, its configuration is represented as a tree rather than a single entry. The top-level entry is the one that contains the `ds-cfg-backend` objectclass, but it also includes the `ds-cfg-je-backend` objectclass that subclasses the `ds-cfg-backend` objectclass and defines the following additional attributes:

- `ds-cfg-backend-deadlock-retry-limit` -- This specifies the number of times that the server should retry an attempted operation in the backend if a deadlock results from two concurrent requests that interfere with each other in a conflicting manner.

- `ds-cfg-backend-directory` -- This specifies the path to the filesystem directory that will be used to hold the Berkeley DB Java Edition database files containing the data for this backend. It may be either an absolute path or a relative path (relative to the directory containing the base of the OpenDS Directory Server installation).

- `ds-cfg-backend-entries-compressed` -- This indicates whether the backend should attempt to compress entries before storing them in the database. Note that this applies only to the entries themselves and does not impact the index data. Further, the effectiveness of the compression will be based on the type of data contained in the entry.

- `ds-cfg-backend-import-buffer-size` -- This specifies the amount of memory that should be used as an internal buffer for index information when processing an LDIF import. The value should be specified as an integer followed by a space and one of the following units:
  - KB -- Indicates that the value is in kilobytes (i.e., should be multiplied by 1000 bytes).
  - KiB -- Indicates that the value is in kibibytes (i.e., should be multiplied by 1024 bytes).
  - MB -- Indicates that the value is in megabytes (i.e., should be multiplied by 1000000 bytes).
  - MiB -- Indicates that the value is in mebibytes (i.e., should be multiplied by 1048576 bytes).
  - GB -- Indicates that the value is in gigabytes (i.e., should be multiplied by 1000000000 bytes).
  - GiB -- Indicates that the value is in gibibytes (i.e., should be multiplied by 1073741824 bytes).

- `ds-cfg-backend-import-pass-size` -- This specifies the maximum number of entries that should be imported in each import pass. An import pass consists of the processing required to import a set of entries as well as the index post-processing required to index those entries. A value of zero for this configuration attribute indicates that all entries should be processed in a single pass, which is the recommended configuration for most deployments, although a nonzero value may be required when importing a very large number of entries if the amount of memory required for index post-processing exceeds the total amount available to the server.

- `ds-cfg-backend-import-queue-size` -- This specifies the size (in number of entries) of the queue that will be used to hold the entries read during an LDIF import.

- `ds-cfg-backend-import-temp-directory` -- This specifies the location of the directory that will be used for the files used to hold temporary information that will be used during the index post-processing phase of an LDIF import. This directory will only be used while an import is in progress and the files created in this directory will be deleted as they are processed.

- `ds-cfg-backend-import-thread-count` -- This specifies the number of threads that will be used for concurrent processing during an LDIF import.

- `ds-cfg-backend-index-entry-limit` -- This specifies the maximum number of entries that will be allowed to match a given index key before that particular index key is no longer maintained (i.e., it is analogous to the ALL IDs threshold in the Sun Java System Directory Server). Note that this is the default limit for the backend, and it may be overridden on a per-attribute basis.

- `ds-cfg-backend-preload-time-limit` -- This specifies the length of time that the backend will be allowed to spend "pre-loading" data when it is initialized. This process may be used to pre-populate the database cache so that it can be more quickly available when the server is processing requests. The value should be specified as an integer followed by a space and one of the following units:
  - ms -- Indicates that the value is in milliseconds.
  - s -- Indicates that the value is in seconds.
  - m -- Indicates that the value is in minutes.

- `ds-cfg-backend-subtree-delete-size-limit` -- This specifies the maximum number of entries that may be deleted from the backend when using the subtree delete control. If a subtree delete operation targets a subtree with more than this number of entries, then multiple passes may be required to remove all entries in that subtree.

In addition to storing the entries themselves, the JE backend also provides an indexing facility that can be used to store information that makes it possible to locate entries very quickly when processing search operations. Indexing is performed on a per-attribute level and different types of indexing may be performed for different kinds of attributes based on how they are expected to be accessed during search operations. The index configuration for each indexed attribute is defined in a separate entry, and each of these entries resides below the entry with an RDN of "cn=Index" immediately below the parent `ds-cfg-je-backend` entry. Each of these index configuration entries should contain the `ds-cfg-je-index` objectclass, which includes the following configuration attributes:

- `ds-cfg-index-attribute` -- This specifies the name of the attribute for which the index is to be maintained.

- `ds-cfg-index-entry-limit` -- This specifies the maximum number of entries that will be allowed to match any index key before the ID list associated with that key is no longer maintained. This will override the value of the `ds-cfg-backend-index-entry-limit` configuration attribute in the base JE backend configuration entry.

- `ds-cfg-index-type` -- This specifies the types of indexing that should be performed for the associated attribute. This is a multivalued attribute, and it may have one or more of the following values:
  - equality -- This index type will be used to help improve the efficiency of searches using equality search filters.
  - ordering -- This index type will be used to help improve the efficiency of searches using "greater than or equal to" or "less than or equal to" search filters.
  - presence -- This index type will be used to help improve the efficiency of searches using presence search filters.
  - substring -- This index type will be used to help improve the efficiency of searches using substring search filters.

As noted above, this backend stores all of its information in a Berkeley DB Java Edition database. This database provides a number of configuration options of its own, and many of those are exposed in the entry with an RDN of "cn=JE Database" that resides immediately below the parent `ds-cfg-je-backend` entry. This entry has an objectclass of `ds-cfg-je-database` and may include the following configuration attributes:

- `ds-cfg-database-cache-percent` -- This specifies the percentage of memory available to the JVM that should be used for caching database contents. The value of this configuration attribute must be an integer between 1 and 90, inclusive. Note that this will only be used if the value of the `ds-cfg-database-cache-size` attribute is set to "0 MB". Otherwise, the value of that configuration attribute will be used instead to control the cache size configuration.

- `ds-cfg-database-cache-size` -- This specifies the amount of memory that should be used for caching database contents. A value of "0 MB" indicates that the `ds-cfg-database-cache-percent` attribute should be used instead to specify the cache size. If this configuration attribute is to be used instead of the `ds-cfg-database-cache-percent` attribute, then its value should be an integer followed by a space and one of the following units:
  - KB -- Indicates that the value is in kilobytes (i.e., should be multiplied by 1000 bytes).
  - KiB -- Indicates that the value is in kibibytes (i.e., should be multiplied by 1024 bytes).
  - MB -- Indicates that the value is in megabytes (i.e., should be multiplied by 1000000 bytes).
  - MiB -- Indicates that the value is in mebibytes (i.e., should be multiplied by 1048576 bytes).

  - GB -- Indicates that the value is in gigabytes (i.e., should be multiplied by 1000000000 bytes).
  - GiB -- Indicates that the value is in gibibytes (i.e., should be multiplied by 1073741824 bytes).

- `ds-cfg-database-checkpointer-bytes-interval` -- This specifies the maximum number of bytes that may be written to the database before it will be forced to perform a checkpoint. This can be used to bound the recovery time that may be required if the database environment is opened without having been properly closed. The value of this configuration attribute should be an integer followed by a space and one of the following units:
  - KB -- Indicates that the value is in kilobytes (i.e., should be multiplied by 1000 bytes).
  - KiB -- Indicates that the value is in kibibytes (i.e., should be multiplied by 1024 bytes).
  - MB -- Indicates that the value is in megabytes (i.e., should be multiplied by 1000000 bytes).
  - MiB -- Indicates that the value is in mebibytes (i.e., should be multiplied by 1048576 bytes).
  - GB -- Indicates that the value is in gigabytes (i.e., should be multiplied by 1000000000 bytes).
  - GiB -- Indicates that the value is in gibibytes (i.e., should be multiplied by 1073741824 bytes).

- `ds-cfg-database-checkpointer-wakeup-interval` -- This specifies the maximum length of time that may pass between checkpoints. Note that this will only be used if the value of the `ds-cfg-database-checkpointer-bytes-interval` configuration attribute is set to "0 MB". If it is used, then the value should be a positive integer followed by a space and one of the following units:
  - us -- Indicates that the value is in microseconds.
  - ms -- Indicates that the value is in milliseconds.
  - s -- Indicates that the value is in seconds.
  - m -- Indicates that the value is in minutes.

- `ds-cfg-database-cleaner-min-utilization` -- This specifies the minimum percentage of "live" data that the database cleaner will attempt to keep in database log files. If the amount of live data in any database log file drops below this percentage, then the cleaner will move the remaining live data in that file to the end of the database and will delete the original file in order to keep the database relatively compact.

- `ds-cfg-database-evictor-lru-only` -- This indicates whether the database should evict existing data from the cache based on an LRU policy (where the least recently used information will be evicted first). If the value of this configuration attribute is set to "false", then eviction will prefer to keep internal nodes of the underlying Btree in the cache over leaf notes, even if the leaf nodes have been accessed more recently, which may be a better configuration for databases in which only a very small portion of the data is cached.

- `ds-cfg-database-evictor-nodes-per-scan` -- This specifies the number of Btree nodes that should be evicted from the cache in a single pass if it is determined that it is necessary to free existing data in order to make room for new information. The value must be an integer between 1 and 1000.

- `ds-cfg-database-lock-num-lock-tables` -- This specifies the number of lock tables that should be used by the underlying database. This can be particularly important to help improve scalability by avoiding contention on systems with large numbers of CPUs. The value of this configuration attribute should be set to a prime number that is less than or equal to the number of worker threads configured for use in the server.

- `ds-cfg-database-log-file-max` -- This specifies the maximum size that may be used for a database log file. The log file size must be between one megabyte and four gigabytes and should be specified as an integer followed by a space and one of the following units:
  - KB -- Indicates that the value is in kilobytes (i.e., should be multiplied by 1000 bytes).
  - KiB -- Indicates that the value is in kibibytes (i.e., should be multiplied by 1024 bytes).
  - MB -- Indicates that the value is in megabytes (i.e., should be multiplied by 1000000 bytes).
  - MiB -- Indicates that the value is in mebibytes (i.e., should be multiplied by 1048576 bytes).
  - GB -- Indicates that the value is in gigabytes (i.e., should be multiplied by 1000000000 bytes).
  - GiB -- Indicates that the value is in gibibytes (i.e., should be multiplied by 1073741824 bytes).

- `ds-cfg-database-logging-file-handler-on` -- This indicates whether the database should maintain a `je.info` file in the same directory as the database log directory. This information will contain information about the internal processing performed by the underlying database.

- `ds-cfg-database-logging-level` -- This specifies the log level that should be used by the database when it is writing information into the `je.info` file. The value should be one of the following strings (in increasing order of the amount of data that may be written to the file):
  - OFF
  - SEVERE
  - WARNING
  - INFO
  - CONFIG
  - FINE
  - FINER
  - FINEST
  - ALL

- `ds-cfg-database-run-cleaner` -- This indicates whether the database cleaner thread should be enabled. The cleaner thread will be used to periodically compact the database by identifying database files with a low (i.e., less than the amount specified by the `ds-cfg-database-cleaner-min-utilization` configuration attribute) percentage of live data, moving the remaining live data to the end of the log and deleting that file.

- `ds-cfg-database-txn-no-sync` -- This indicates whether database writes should be primarily written to an internal buffer but not immediately written to disk. Setting the value of this configuration attribute to "true" may improve write performance but could cause some number of the most recent changes to be lost if the OpenDS Directory Server or the underlying JVM exits abnormally, or if an OS or hardware failure occurs (a behavior similar to running with transaction durability disabled in the Sun Java System Directory Server).

- `ds-cfg-database-txn-write-no-sync` -- This indicates whether the database should synchronously flush data as it is written to disk. If this value is set to "false", then all data written to disk will be synchronously flushed to persistent storage and thereby providing full durability. If it is set to "true", then data may be cached for a period of time by the underlying operating system before actually being written to disk. This may improve performance, but could cause some number of the most recent changes to be lost in the event of an underlying OS or hardware failure (but not in the case that the OpenDS Directory Server or the JVM exits abnormally).

The Backup Backend

The backup backend provides read-only access to the set of backups that are available for the OpenDS Directory Server. It is provided as a convenience feature that makes it easier to determine what backups are available to be restored if necessary. The `org.opends.server.backends.BackupBackend` class provides the implementation for this backend.

This is a relatively simple backend that only needs a single entry to provide its configuration. That entry contains the `ds-cfg-backup-backend` objectclass, which includes the following attributes:

- `ds-cfg-backup-directory` -- This specifies the path to a backup directory containing one or more backups for a particular backend. It may be either an absolute path or one that is relative to the base of the OpenDS Directory Server installation. This is a multivalued attribute, and each value may specify a different backup directory if desired (e.g., one for each backend for which backups are taken).

The Configuration Backend

The configuration backend provides access to the OpenDS Directory Server configuration through a part of the directory hierarchy. This configuration entry exists primarily to ensure that the backend is properly registered with the server for operational purposes, and it should not be altered or removed. Certain functionality (e.g., the ability to back up the configuration) may be lost if this configuration entry is modified.

The Monitor Backend

The monitor backend provides access to the OpenDS Directory Server monitor information. This includes the information collected by the various monitor providers registered with the server (see the Monitor Provider Configuration section for additional details about monitor providers). The `org.opends.server.backends.MonitorBackend` class provides the implementation for this backend.

This is a relatively simple backend in which only read operations will be allowed, and in which the information that it provides is not actually stored anywhere but gets constructed on the fly when it is requested. There is no additional configuration information that is needed for this backend to function beyond that which is provided in the `ds-cfg-backend` objectclass. However, if any additional attributes (along with corresponding objectclasses) are added to this configuration entry other than those defined in the `ds-cfg-backend` objectclass, then those attributes and values will appear in the generated cn=monitor entry.

The Schema Backend

The schema backend provides access to the OpenDS Directory Server schema information, including the attribute types, objectclasses, attribute syntaxes, matching rules, matching rule uses, DIT content rules, and DIT structure rules that it contains. At the present time, this backend is read-only but in the future it will be enhanced to allow updates to the schema configuration while the server is running. The `org.opends.server.backends.SchemaBackend` class provides the implementation for this backend.

The configuration entry for this backend is based on the `ds-cfg-schema-backend` structural objectclass, which includes the following attributes:

- `ds-cfg-schema-entry-dn` -- This defines the base DN(s) at which the schema information will be published, in addition to the value included in the `ds-cfg-backend-base-dn` configuration attribute. The value provided in the `ds-cfg-backend-base-dn` configuration attribute is the only one that will appear in the `subschemaSubentry` operational attribute of the server's root DSE (which is necessary because that is a single-valued attribute), but the `ds-cfg-schema-entry-dn` attribute may be used to make the schema information available in other locations as well in case certain client applications have been hard-coded to expect the schema to reside in a specific location.

The Task Backend

The task backend provides a mechanism for processing tasks in the OpenDS Directory Server. Tasks are intended to provide access to certain types of administrative functions in the server that may not otherwise be convenient to perform remotely. Tasks that are currently available for use provide the ability to backup and restore backends, to import and export LDIF files, and to stop and restart the server. The details of the task to perform are held in an entry that is added below the root of the task backend, and then the task backend is responsible for decoding that task entry and ensuring that it is processed as requested. Tasks may be invoked immediately, but they may also be scheduled for execution at some future time. It is also expected that task backend will be given the ability process recurring tasks, which can be used to help ensure that maintenance operations (e.g., backups) are performed automatically on a regular basis. The `org.opends.server.backends.task.TaskBackend` class provides the entry point for the task backend implementation.

The configuration entry for this backend is based on the `ds-cfg-task-backend` structural objectclass, which includes the following attributes:

- `ds-cfg-task-backing-file` -- This specifies the path to the backing file for storing information about the tasks configured in the server. It may be either an absolute path or a path that is relative to the base of the OpenDS Directory Server instance.

- `ds-cfg-task-retention-time` -- This specifies the length of time that task entries should be retained after processing on the associated task has been completed. The value of this configuration attribute should be specified as an integer followed by a space and one of the following units:
  - seconds
  - minutes
  - hours
  - days
  - weeks

The Connection Handler Configuration

The "cn=Connection Handlers,cn=config" configuration entry is the parent entry for all connection handlers defined in the OpenDS Directory Server. Connection handlers are responsible for handling all interaction with the clients, including accepting the connections, reading requests, and sending responses.

All connection handler configuration entries must contain the `ds-cfg-connection-handler` structural objectclass, which defines the following configuration attributes:

- `ds-cfg-connection-handler-class` -- This specifies the fully-qualified name of the Java class that provides the connection handler implementation. This class must be a concrete subclass of the `org.opends.server.api.ConnectionHandler` superclass.

- `ds-cfg-connection-handler-enabled` -- This indicates whether the connection handler is enabled for use (i.e., whether it should be listening for new client connections).

There are multiple types of connection handlers defined in the OpenDS Directory Server, and they will be described individually in the remainder of this section.

The LDAP Connection Handler

The LDAP connection handler is used to interact with clients using LDAP. In particular, it provides full support for LDAPv3 and limited support for LDAPv2. The implementation for this connection handler is defined in the `org.opends.server.protocols.ldap.LDAPConnectionHandler` class. The associated configuration entry is based on the `ds-cfg-ldap-connection-handler` objectclass, which includes the following configuration attributes:

- `ds-cfg-accept-backlog` -- This specifies the maximum number of pending connection attempts that will be allowed to queue up in the accept backlog before the server starts rejecting new connection attempts. This is primarily an issue for cases in which a large number of connections are established to the server in a very short period of time (e.g., a benchmark utility that creates a large number of client threads that each have their own connection to the server) and the connection handler is unable to keep up with the rate at which the new connections are established.

- `ds-cfg-allow-ldapv2` -- This indicates whether connections from LDAPv2 clients will be allowed. If LDAPv2 clients will be allowed, then only a minimal degree of special support will be provided for them to ensure that LDAPv3-specific protocol elements (e.g., controls, extended response messages, intermediate response messages, referrals, etc.) are not sent to an LDAPv2 client.

- `ds-cfg-allow-start-tls` -- This indicates whether clients will be allowed to use the StartTLS extended operation to initiate secure communication over an otherwise insecure channel. Note that this will only be allowed if the connection handler is not configured to use SSL, and if the server is configured with a valid key manager provider and a valid trust manager provider.

- `ds-cfg-allow-tcp-reuse-address` -- This indicates whether the `SO_REUSEADDR` socket option will be used on the server listen socket to potentially allow the reuse of socket descriptors for clients in a `TIME_WAIT` state. This may help the server avoid temporarily running out of socket descriptors in cases in which a very large number of short-lived connections have been established from the same client system.

- `ds-cfg-keep-stats` -- This indicates whether the connection handler should keep statistics about the number and types of operations requested over LDAP and the amount of data sent and received.

- `ds-cfg-listen-address` -- This specifies the address or set of addresses on which this connection handler should listen for connections from LDAP clients. Multiple addresses may be provided as separate values for this attribute. If no values are provided, then the connection handler will listen on all interfaces.

- `ds-cfg-listen-port` -- This specifies the port on which this connection handler should listen for requests from LDAP clients. Only a single listen port may be provided for each instance of the LDAP connection handler.

- `ds-cfg-max-request-size` -- This specifies the size of the largest LDAP request message that will be allowed by this connection handler (it is analogous to the maxBERSize configuration attribute of the Sun Java System Directory Server). This can help prevent denial-of-service attacks by clients that indicate they will send extremely large requests to the server causing it to attempt to allocate large amounts of memory. The value of this configuration attribute is specified as an integer followed by a space and one of the following units:
  - B -- Indicates that the value is in bytes.
  - KB -- Indicates that the value is in kilobytes (i.e., should be multiplied by 1000 bytes).
  - KiB -- Indicates that the value is in kibibytes (i.e., should be multiplied by 1024 bytes).
  - MB -- Indicates that the value is in megabytes (i.e., should be multiplied by 1000000 bytes).
  - MiB -- Indicates that the value is in mebibytes (i.e., should be multiplied by 1048576 bytes).

- `ds-cfg-num-request-handlers` -- This specifies the number of request handlers that will be used to read requests from clients. The LDAP connection handler uses one thread to accept new connections from clients, but uses one or more additional threads to read requests from existing client connections. This can help ensure that new requests are read efficiently and that the connection handler itself does not become a bottleneck when the server is under heavy load from many clients at the same time.

- `ds-cfg-send-rejection-notice` -- This indicates whether the connection handler should send a notice of disconnection extended response message to the client if a new connection is rejected for some reason. This message may provide an explanation indicating the reason that the connection was rejected.

- `ds-cfg-ssl-cert-nickname` -- This specifies the nickname (also called the alias) of the certificate that the connection handler should use when performing SSL communication. This is only applicable if `ds-cfg-use-ssl` has a value of "true".

- `ds-cfg-ssl-client-auth-policy` -- This specifies the policy that the connection handler should use regarding client SSL certificates. This is only applicable if `ds-cfg-use-ssl` has a value of "true". The value must be one of the following strings:
  - disabled -- The client listener will not request that clients provide their own certificates when performing SSL negotiation.
  - optional -- The client listener will request that clients provide their own certificates when performing SSL negotiation, but will still accept the connection even if the client does not provide a certificate.
  - required -- The client listener will request that clients provide their own certificates when performing SSL negotiation, and will refuse access to any client that does not provide a certificate.

- `ds-cfg-use-ssl` -- This indicates whether the connection handler should use SSL to encrypt communication with the clients.

- `ds-cfg-use-tcp-keepalive` -- This indicates whether the connection handler should use the `SO_KEEPALIVE` socket option to indicate that TCP keepalive messages should periodically be sent to the client to verify that the associated connection is still valid. This may also help prevent cases in which intermediate network hardware could silently drop an otherwise idle client connection, provided that the keepalive interval configured in the underlying operating system is smaller than the timeout enforced by the network hardware.

- `ds-cfg-use-tcp-nodelay` -- This indicates whether the connection handler should use the `TCP_NODELAY` socket option to ensure that response messages to the client are sent immediately rather than potentially waiting to determine whether additional response messages can be sent in the same packet. In most cases, using the `TCP_NODELAY` socket option will provide better performance and lower response times, but disabling it may help for some cases in which the server will send a large number of entries to a client in response to a search request.

The JMX Connection Handler

The JMX connection handler is used to interact with clients using the Java Management Extensions (JMX) protocol. The implementation for this connection handler is defined in the `org.opends.server.protocols.jmx.JmxConnectionHandler` class. The associated configuration entry is based on the `ds-cfg-jmx-connection-handler` objectclass, which includes the following configuration attributes:

- `ds-cfg-listen-port` -- This specifies the port number on which the connection handler will listen for connections from clients. Only a single port number may be provided, and the connection handler will listen on all interfaces.

- `ds-cfg-ssl-cert-nickname` -- This specifies the nickname (also called the alias) of the certificate that the connection handler should use when performing SSL communication. This is only applicable if `ds-cfg-use-ssl` has a value of "true".

- `ds-cfg-use-ssl` -- This indicates whether the connection handler should use SSL to encrypt communication with the clients.

The Entry Cache Configuration

The "cn=Entry Cache,cn=config" configuration entry holds the configuration for the OpenDS entry cache. Entry caching in the OpenDS Directory Server is significantly different from entry caching in the Sun Java System Directory Server in a number of ways, including:

- The OpenDS Directory Server has only a single entry cache, regardless of the number of backends, whereas the Sun Java System Directory Server uses a separate entry cache per backend. Using a single cache makes management of the server easier, and also makes it possible to ensure that the available memory resources are used as efficiently as possible.

- The OpenDS Directory Server uses a "look aside" caching mechanism (in which it is up to the backend to determine whether to interact with the cache for any given operation), whereas the Sun Java System Directory Server uses a "look through" cache in which all interaction with the backends goes through the cache. Using a "look aside" cache makes it possible to have better control over what goes into the cache, and can help avoid "poisoning" the cache with operations like unindexed searches need to examine a large number of entries that are not likely to be re-accessed in the short term.

- The mechanism used to provide the entry cache is extensible, so that different caching mechanisms can easily be configured for use in the server. This makes it possible to tailor the behavior of the cache to the needs of the underlying directory.

- The entry cache can be easily disabled in the OpenDS Directory Server and with fewer adverse effects than doing so in the Sun Java System Directory Server. In fact, the OpenDS Directory Server has been designed so that it can operate efficiently without the need for an entry cache in most cases, which allows much more efficient use of the available memory resources on the system.

All entry cache configuration entries must contain the `ds-cfg-entry-cache` structural objectclass, which defines the following configuration attributes:

- `ds-cfg-entry-cache-class` -- This specifies the fully-qualified name of the Java class that provides the entry cache implementation. This must be a concrete subclass of the `org.opends.server.api.EntryCache` superclass.

- `ds-cfg-entry-cache-enabled` -- This indicates whether the entry cache will actually be enabled for use in the server.

There are multiple entry cache implementations in the OpenDS Directory Server, and they will be described in the remainder of this section.

The Default Entry Cache

As its name implies, this is the default entry cache that will be used in the server if no other cache is configured, and it does not actually store any entries. That is, it ignores any request to store entries in the cache, and any request to retrieve an entry from the cache receives an indication that the specified entry is not contained in the cache. This implementation is contained in the `org.opends.server.extensions.DefaultEntryCache` class, and does not require any additional configuration.

The FIFO Entry Cache

The FIFO entry cache essentially creates a first-in, first-out list that may be used to hold entries that have been accessed in the server. Because it is a FIFO, the entries that have been in the cache the longest will be the ones to get removed if it is necessary to make more space for new entries. This is different from the behavior used by the Sun Java System Directory Server, which is based on an LRU (least recently used) algorithm. The use of a FIFO does mean that "hot" entries are more likely to eventually be pushed out of the cache than with an LRU, but it also requires significantly less locking and may allow for higher overall performance and better scalability.

This implementation also introduces new capabilities into the entry cache that have been made possible as a result of our look aside approach. In particular, it is possible to control what types of entries are held in the cache through the use of include and/or exclude filters. If it is expected that certain types of entries will be the most commonly accessed, then the cache can be configured to only include them. Alternately, if there are certain types of entries that are unlikely be accessed repeatedly, then the cache can be configured to exclude them.

Another interesting difference between this entry cache implementation and that of the Sun Java System Directory Server is in the area of memory consumption. In the Sun Java System Directory Server, the size of the entry cache is generally configured in terms of the amount of memory that it should be allowed to use. This requires the ability to determine the size of any individual entry, which is particularly difficult to achieve with any accuracy in Java. Further, the JVM enforces an upper bound on the amount of memory that can be used (which is configured when the JVM is started), so it does not allow applications to have direct control over the amount of memory in use. It is possible for an application to determine the maximum amount of memory that is available to the JVM, and also the amount of memory that is currently in use (although this may include garbage that has not yet been collected). The FIFO entry cache implementation will use this information in order to maintain a maximum percentage of memory consumption. That is, instead of attempting to keep track of how much memory is in use by the entries in the cache, it will keep track of how much memory is free in the JVM and will always tree to keep some minimum free percentage available.

The FIFO entry cache implementation is contained in the `org.opends.server.extensions.FIFOEntryCache` class, and its configuration is held in the `ds-cfg-fifo-entry-cache` structural objectclass which includes the following attributes:

- `ds-cfg-exclude-filter` -- This specifies a set of search filters that will be compared against entries to determine whether they should be excluded from the cache. This is a multivalued attribute, and if an entry matches any of the filters listed then it will not be stored in the cache.

- `ds-cfg-include-filter` -- This specifies a set of search filters that will be compared against entries to determine whether they should be included in the cache. This is a multivalued attribute, and entries will not be stored in the cache unless they match at least one of these filters.

- `ds-cfg-lock-timeout` -- This specifies the maximum length of time that the entry cache should allow a thread to block while attempting to acquire a lock on the target entry. This should be specified as an integer followed by a space and one of the following units:
  - ms -- Indicates that the value is in milliseconds
  - s -- Indicates that the value is in seconds

- `ds-cfg-max-entries` -- This specifies the maximum number of entries that should be maintained in the cache at any given time. If an attempt is made to store a new entry in the cache when it is already full, it will cause the oldest entry to be removed.

- `ds-cfg-max-memory-percent` -- This specifies the maximum percentage of memory that the entry cache will attempt to have utilized at any given time. That is, the entry cache will try to keep around (100 - `ds-cfg-max-memory-percent`) percent of the JVM memory free.

The Soft Reference Entry Cache

The soft reference entry cache provides an alternate memory management implementation. As noted in the above discussion of the FIFO entry cache, memory management within the JVM is an interesting prospect because of the somewhat limited amount of information available. While the FIFO entry cache attempts to monitor memory consumption based on the information that is available, the soft reference entry cache takes a different approach by using soft references to manage the data. A soft reference is a way to keep track of an object in a manner that does not prevent it from being garbage collected if the JVM determines that it needs more memory for some reason. In this case, the soft reference cache doesn't do any memory itself, but rather allows the JVM to decide when it needs more memory and will allow it to get rid of the entries as necessary.

The soft reference entry cache is implemented in the `org.opends.server.extensions.SoftReferenceEntryCache` class, and includes the following configuration attributes in the `ds-cfg-soft-reference-entry-cache` objectclass:

- `ds-cfg-exclude-filter` -- This specifies a set of search filters that will be compared against entries to determine whether they should be excluded from the cache. This is a multivalued attribute, and if an entry matches any of the filters listed then it will not be stored in the cache.

- `ds-cfg-include-filter` -- This specifies a set of search filters that will be compared against entries to determine whether they should be included in the cache. This is a multivalued attribute, and entries will not be stored in the cache unless they match at least one of these filters.

- `ds-cfg-lock-timeout` -- This specifies the maximum length of time that the entry cache should allow a thread to block while attempting to acquire a lock on the target entry. This should be specified as an integer followed by a space and one of the following units:
  - ms -- Indicates that the value is in milliseconds
  - s -- Indicates that the value is in seconds

The Extended Operation Handler Configuration

The "cn=Extended Operation Handlers,cn=config" configuration entry is the parent for all extended operation handlers defined in the OpenDS Directory Server. Extended operation handlers are responsible for all processing different types of extended operations in the server.

All extended operation handler configuration entries must contain the `ds-cfg-extended-operation-handler` structural objectclass, which defines the following configuration attributes:

- `ds-cfg-extended-operation-handler-class` -- This specifies the fully-qualified name of the Java class that provides the extended oepration handler implementation. This class must be a concrete subclass of the `org.opends.server.api.ExtendedOperationHandler` superclass.

- `ds-cfg-extended-operation-handler-enabled` -- This indicates whether the extended operation handler is enabled for use (i.e., whether the types of extended operations that it defines will be allowed in the server).

There are multiple types of extended operation handlers defined in the OpenDS Directory Server, and they will be described individually in the remainder of this section.

The Cancel Extended Operation Handler

The cancel extended operation is defined in RFC 3909 and allows clients to cancel an operation that has been previously submitted on the same connection. It is similar to the standard LDAP abandon operation, with the primary exception that both the cancel request itself and the operation being cancelled receive a response.

The cancel extended operation handler is implemented in the `org.opends.server.extensions.CancelExtendedOperation` class. It does not require any custom configuration.

The Password Modify Extended Operation Handler

The password modify extended operation is defined in RFC 3062 and provides the ability for end users to change their own passwords, or for administrators to reset user passwords. It includes the ability for users to provide their current password for further confirmation of their identity when changing the password, and it also includes the ability to generate a new password if the user doesn't provide one.

The password modify extended operation handler is implemented in the `org.opends.server.extensions.PasswordModifyExtendedOperation` class. It does not require any custom configuration.

The StartTLS Extended Operation Handler

The StartTLS extended operation is defined in RFC 2830 and has been updated in RFC 4513. It provides the ability for a client to initiate a TLS-encrypted channel over an existing insecure connection. This makes it possible for clients to communicate in a secure manner with the server over the same port that is also used for clear-text communication.

The StartTLS extended operation handler is implemented in the `org.opends.server.extensions.StartTLSExtendedOperation` class. It does not require any custom configuration.

The "Who Am I?" Extended Operation Handler

The "Who Am I?" extended operation is defined in RFC 4532. It provides the client to determine the authorization identity for a client, which can be useful if the client wishes to confirm that it is authenticated as the expected user, or to determine the identity of the client if it authenticated using a mechanism like SASL EXTERNAL in which the client may not know its identity in the directory.

The "Who Am I?" extended operation handler is implemented in the `org.opends.server.extensions.WhoAmIExtendedOperation` class. It does not require any custom configuration.

The Identity Mapper Configuration

The "cn=Identity Mappers,cn=config" configuration entry is the parent for all identity mappers defined in the OpenDS Directory Server. Identity mappers are responsible for identifying a user based on a provided username, principal, or other information that might have been provided by the client. Identity mappers are used in the processing of several SASL mechanisms to map an authorization ID (e.g., a Kerberos principal when using GSSAPI) to a directory user, and they are also used when processing requests with the proxied authorization control.

All identity mapper configuration entries must contain the `ds-cfg-identity-mapper` structural objectclass, which defines the following configuration attributes:

- `ds-cfg-identity-mapper-class` -- This specifies the fully-qualified name of the Java class that provides the identity mapper logic. This must be a concrete subclass of the `org.opends.server.api.IdentityMapper` superclass.

- `ds-cfg-identity-mapper-enabled` -- This indicates whether the identity mapper is enabled for use in the server.

There can be multiple types of identity mappers defined in the OpenDS Directory Server, and they will be described individually in the remainder of this section.

The Exact Match Identity Mapper

The exact match identity mapper is a very simple implementation that simply looks for exactly one user whose entry contains the exact authentication ID or authorization ID value provided by the client in a specified attribute (e.g., the username provided by the client for DIGEST-MD5 authentication must match the value of the `uid` attribute).

The exact match identity mapper is implemented in the `org.opends.server.extensions.ExactMatchIdentityMapper`. The `ds-cfg-exact-match-identity-mapper` objectclass includes the following attributes that should be used to configure it:

- `ds-cfg-match-attribute` -- This specifies the name of the attribute whose value must exactly equal the provided authentication or authorization ID. The specified attribute should be indexed for equality in all appropriate backends. This is a multivalued attribute, and if multiple values are given then the provided value may be contained in any of them (i.e., the internal search performed will include a logical OR across all of these values).

- `ds-cfg-match-base-dn` -- This specifies the search base DN that will be used when searching for the entry with the specified authentication or authorization ID value. This is a multivalued attribute, and if multiple values are given then searches will be performed below all specified base DNs. If no values are given, then the search will be performed as if the base specified had been the root DSE.

The Logger Configuration

The "cn=Loggers,cn=config" configuration entry is the parent for all loggers defined in the OpenDS Directory Server. There are three primary types of loggers that may be used:

- Access Loggers -- These are used to log information about the types of operations processed by the server. All access loggers must be concrete subclasses of the `org.opends.server.api.AccessLogger` class, and their associated configuration entries must contain the `ds-cfg-access-logger` structural objectclass.

- Error Loggers -- These are used to log information about any warnings, errors, or significant events that are encountered during server processing. All error loggers must be concrete subclasses of the `org.opends.server.api.ErrorLogger` class, and their associated configuration entries must contain the `ds-cfg-error-logger` structural objectclass.

- Debug Loggers -- These are used to log information that may be used for debugging or troubleshooting problems in the server, or for providing more detailed information about the processing that it performs. All debug loggers must be concrete subclasses of the `org.opends.server.api.DebugLogger` class, and their associated configuration entries must contain the `ds-cfg-debug-logger` structural objectclass.

Each of those objectclasses inherits from the common ds-cfg-logger structural objectclass, which defines the following attributes:

- `ds-cfg-logger-class` -- This specifies the fully-qualified name of the Java class that provides the logger implementation.

- `ds-cfg-logger-enabled` -- This indicates whether the access logger will be enabled for use.

It should be pointed out that there are significant differences in the logger architecture for the OpenDS Directory Server than for the Sun Java System Directory Server. The most notable differences include:

- The OpenDS Directory Server allows for any number of loggers of any given type to be defined and active at any given time. This makes it possible to log to different locations or different types of repositories, and also to potentially have different sets of criteria for what to include in the logs (e.g., one access log may hold everything, one may hold just operations with a nonzero result code, one may hold just write operations, etc.).

- The Sun Java System Directory Server defines another type of logger called the audit logger which is used to hold information about the changes that are made to directory data.

In the OpenDS Directory Server, the audit logging capability is still present but it is classified as a type of access logger.

As noted above, there are different superclasses and different structural objectclasses for the different types of loggers. None of those objectclasses themselves include any configuration attributes, but other objectclasses that inherit from them do define a number of configuration options.

It should be noted that the current OpenDS logger implementation exhibits quite poor performance, and therefore will need to be rewritten in a more efficient manner. Further, the current configuration does not provide a mechanism that can be used to easily define filters to restrict the types of content to include in the log. As such, the existing implementation and its associated configuration will likely change significantly in the future. However, the details of the current configuration will be provided here for completeness.

There are some common configuration attributes that are shared by all loggers currently in use. They include:

- `ds-cfg-log-file` -- This specifies the path to the log file to be written. It may be either an absolute path, or it may be relative to the install root of the OpenDS instance.

- `ds-cfg-buffer-size` -- This specifies the size in bytes for the log file buffer. By default, log information will be written to an internal buffer and flushed when the buffer is full. This can provide better performance but could introduce a risk of losing any buffered log information if the JVM or underlying system crashes without flushing the data to disk. Setting the buffer size to zero bytes will disable the buffering capability.

- `ds-cfg-rotation-policy` -- This specifies the policy to use in order to determine when the log file needs to be rotated. This is a multivalued attribute, and any time one of the criteria is met the log file will be rotated. Allowed values for this configuration attribute include:
  - fixedtime -- Indicates that log rotation should occur at regular intervals (i.e., a fixed length of time between rotations).
  - size -- Indicates that log rotation should occur when the log file reaches a specified size.
  - timeofday -- Indicates that log rotation should occur at a specific time of the day.

- `ds-cfg-fixed-time-limit` -- This specifies the maximum length of time in milliseconds that should be allowed to elapse between log file rotations.

- `ds-cfg-size-limit` -- This specifies the maximum size in bytes that a log file should be allowed to consume before it is rotated.

- `ds-cfg-time-of-day` -- This specifies the time of day at which log rotation should occur. It should be specified in a 24-hour HHMM format (e.g., "0000" indicates midnight, and "1830" indicates 6:30 PM).

- `ds-cfg-rotation-action` -- This specifies any additional actions that should be taken when the log file is rotated. Rotation actions that may be used include:
  - GZIP_COMPRESSION -- Indicates that the file should be compressed using the GZIP compression algorithm.
  - ZIP_COMPRESSION -- Indicates that the file should be compressed using the PKZIP compression algorithm.
  - ENCRYPT -- Indicates that the file should be encrypted (this is not currently implemented).
  - SIGN -- Indicates that the file should be cryptographically signed (this is not currently implemented).

- `ds-cfg-retention-policy` -- This specifies the policy to use in order to determine how long to retain log files that have been rotated. This is a multivalued attribute, and any time one of the criteria is met, an archived file will be removed. Allowed values for this configuration attribute include:
  - numberoffiles -- This specifies that there should be an upper limit on the number of archived log files to retain.
  - diskspaceused -- This specifies that there should be an upper limit on the amount of disk space that archived log files should consume.
  - freediskspace -- This specifies that there should be a lower limit on the amount of free disk space that will be required to hold archived log files (this is not currently implemented).

- `ds-cfg-number-of-files` -- This specifies the maximum number of archived log files to retain.

- `ds-cfg-disk-space-used` -- This specifies the maximum amount of disk space (in bytes) that should be used for archived log files.

- `ds-cfg-free-disk-space` -- This specifies the minimum amount of free disk space (in bytes) that should be allowed on the filesystem storing the archived log files.

- `ds-cfg-thread-time-interval` -- This specifies the length of time (in milliseconds) that a background log management thread should sleep between checks to determine whether a time-based rotation (either fixed-interval or time-of-day) is needed, or whether disk space consumption warrants deleting an archived log file.

The access and audit loggers allow an additional configuration attribute that do not apply to the error or debug loggers:

- `ds-cfg-suppress-internal-operations` -- This indicates whether internal operations (e.g., operations initiated by plugins or synchronization) should be logged along with those requested by end users.

The error and debug loggers also allow additional configuration attributes that do not apply to the access loggers:

- `ds-cfg-default-severity` -- This specifies the default severity levels that should be written to the error or debug log. This is a multivalued attribute, as the severities are not hierarchical. For the error logger, the defined severities include:
  - FATAL_ERROR
  - GENERIC_DEBUG
  - INFORMATIONAL
  - MILD_ERROR
  - MILD_WARNING
  - NOTICE
  - SEVERE_ERROR
  - SEVERE_WARNING
  - SHUTDOWN_DEBUG
  - STARTUP_DEBUG

  For the debug logger, the defined severities include:
  - COMMUNICATION
  - ERROR
  - INFO
  - VERBOSE
  - WARNING

- `ds-cfg-override-severity` -- This provides the ability to override the default severity level for messages associated with one or more log categories. This is a multivalued attribute, with the ability to override the default severities for one category per value. Values should be in the form "category=severity1,severity2,..." (e.g., to only use the COMMUNICATION, ERROR, and WARNING severities for the PROTOCOL_READ debug category, the value would be "PROTOCOL_READ=COMMUNICATION,ERROR,WARNING"). The defined error log categories include:
  - ACCESS_CONTROL
  - BACKEND
  - CONFIGURATION
  - CONNECTION_HANDLING
  - CORE_SERVER
  - EXCEPTION
  - EXTENDED_OPERATION
  - EXTENSIONS
  - PLUGIN
  - REQUEST_HANDLING
  - SASL_MECHANISM
  - SCHEMA
  - SHUTDOWN
  - STARTUP

- SYNCHRONIZATION
- TASK.

The defined debug log categories include:

- ACCESS_CONTROL
- BACKEND
- CONFIGURATION
- CONNECTION_HANDLING
- CONSTRUCTOR
- CORE_SERVER
- DATA_READ
- DATA_WRITE
- DATABASE_ACCESS
- DATABASE_READ
- DATABASE_WRITE
- EXCEPTION
- EXTENDED_OPERATION
- EXTENSIONS
- METHOD_ENTER
- PASSWORD_POLICY
- PLUGIN
- PROTOCOL_READ
- PROTOCOL_WRITE
- SASL_MECHANISM
- SCHEMA
- SHUTDOWN
- STARTUP
- SYNCHRONIZATION

The Matching Rule Configuration

The "cn=Matching Rules,cn=config" configuration entry is the parent for all matching rules defined in the OpenDS Directory Server. Matching rules can be used to make comparisons between attribute values and assertion values. There are four basic types of matching rules:

- Equality matching rules are used to determine if two values are logically equal to one another. Equality matching rules are implemented as subclasses of the `org.opends.server.api.EqualityMatchingRule` class and are configured using the `ds-cfg-equality-matching-rule` structural objectclass.

- Substring matching rules are used to determine if an attribute value contains a given substring assertion. Substring matching rules are implemented as subclasses of the `org.opends.server.api.SubstringMatchingRule` class and are configured using the `ds-cfg-substring-matching-rule` structural objectclass.

- Ordering matching rules are used to determine the relative order between two values in a sorted list. Ordering matching rules are implemented as subclasses of the `org.opends.server.api.OrderingMatchingRule` class and are configured using the `ds-cfg-ordering-matching-rule` structural objectclass.

- Approximate matching rules are used to determine whether two values are approximately equal to each other (where the definition of "approximately equal to" is not clearly defined, but in many cases a "sounds like" implementation is used). Approximate matching rules are implemented as subclasses of the `org.opends.server.api.ApproximateMatchingRule` class and are configured using the `ds-cfg-approximate-matching-rule` structural objectclass.

Matching rules are frequently associated with an attribute syntax and are used to compare values according to that syntax. For example, the distinguishedNameEqualityMatch matching rule can be used to determine whether two DNs are equal and can ignore unnecessary spaces around commas and equal signs, differences in capitalization in attribute names, etc.

All of the specific matching rule objectclasses listed above inherit from the `ds-cfg-matching-rule` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-matching-rule-class` -- This specifies the fully-qualified name of the Java class that provides the logic for the matching rule.

- `ds-cfg-matching-rule-enabled` -- This indicates whether the matching rule is enabled for use in the server.

There are a number of matching rules implemented in the OpenDS Directory Server and defined in the server configuration. All of the provided matching rule implementations exist in the `org.opends.server.schema` package. None of them require any custom configuration attributes or objectclasses.

The Monitor Provider Configuration

The "cn=Monitor Providers,cn=config" configuration entry is the parent for all monitor providers defined in the OpenDS Directory Server. Monitor providers are responsible for publishing information about the server that may be useful for monitoring, debugging, or informational purposes. As with the Sun Java System Directory Server, the "cn=monitor" entry exists and contains a basic set of server-wide information, but there are also a number of subordinate entries with specific information about individual server components. Some monitor providers are registered automatically by other components that might be in use (e.g., if the LDAP connection handler is enabled and the `ds-cfg-keep-stats` attribute is set to true, then that connection handler will automatically register a monitor provider to publish that information). Other monitor providers must be explicitly configured for use within the server. In those cases, the configuration entries must contain the `ds-cfg-monitor-provider` structural objectclass which includes the following attributes:

- `ds-cfg-monitor-provider-class` -- This specifies the fully-qualified name of the Java class that provides the monitor provider implementation. This class must be a concrete subclass of the `org.opends.server.api.MonitorProvider` superclass.

- `ds-cfg-monitor-provider-enabled` -- This indicates whether the monitor provider is enabled for use in the server.

There are a number of monitor providers defined in the server configuration, but none of them need any additional configuration attributes or objectclasses. Those monitor providers include:

- JVM Stat Trace -- Provides the ability to retrieve a complete stack trace from all threads that exist in the JVM.

- System Info -- Provides information about the JVM and the underlying system on which the server is running.

- Version -- Provides information about the OpenDS Directory Server version.

The Password Generator Configuration

The "cn=Password Generators,cn=config" configuration entry is the parent for all password generators defined in the OpenDS Directory Server. Password generators can be used to automatically generate new passwords for users through the password modify extended operation. Because the server allows any number of password validators to be defined, which can impose any kinds of restrictions on the characteristics of valid passwords, it is not feasible for the server to attempt to generate a password on its own that will meet all the requirements of all the validators. The password generator API makes it possible to provide custom logic for creating a new password, which will presumably be acceptable for use by the associated end user.

All password generator configuration entries must contain the ds-cfg-password-generator structural objectclass, which defines the following configuration attributes:

- `ds-cfg-password-generator-class` -- This specifies the fully-qualified name of the Java class that provides the password generator implementation. All password generators must be concrete subclasses of the `org.opends.server.api.PasswordGenerator` superclass.

- `ds-cfg-password-generator-enabled` -- This indicates whether the associated password generator is configured for use in the server.

There can be multiple types of password generators defined in the OpenDS Directory Server, and they will be described individually in the remainder of this section.

The Random Password Generator

The random password generator creates passwords from characters selected at random from different character sets. The individual character sets to use when generating the password, as well as the overall format of the value, are available as configurable options. The random password generator is implemented in the `org.opends.server.extensions.RandomPasswordGenerator` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-password-character-set` -- This defines one or more named character sets. This is a multivalued attribute, with each value defining a different character set. The format of the character set is the name of the set followed by a colon and the characters that should be in that set. For example, the value "alpha:abcdefghijklmnopqrstuvwxyz" would define a character set named "alpha" containing all of the lowercase ASCII alphabetic characters.

- `ds-cfg-password-format` -- This specifies the format to use for the generated password. The value is a comma-delimited list of elements in which each of those elements is comprised of the name of a character set defined in the `ds-cfg-password-character-set` attribute, a colon, and the number of characters to include from that set. For example, a value of "alpha:3,numeric:2,alpha:3" would generate an 8-character password in which the first three characters are from the "alpha" set, the next two are from the "numeric" set, and the final three are from the "alpha" set.

The Password Policy Configuration

The "cn=Password Policies,cn=config" configuration entry is the parent for all password policies defined in the OpenDS Directory Server (this is a departure from the Sun Java System Directory Server, in which password policies could be defined virtually anywhere, including in the user data). Password policies define a number of password management rules, as well as requirements for authentication processing.

The password policy options available in the OpenDS Directory Server come from three primary sources. Where appropriate, the behavior detailed in the "Password Policy for LDAP Directories" Internet Draft is used, particularly for operational attributes in the user entry that maintain password policy state information. Additional features not defined in the draft but that have historically been available in the Sun Java System Directory Server are also included, with consideration given to the migration process. Finally, a number of new features not previously available have been included in the OpenDS Directory Server implementation.

Password policy configuration entries must contain the `ds-cfg-password-policy` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-account-status-notification-handler-dn` -- This specifies the DN(s) of the configuration entries for any account status notification handler(s) that should be used for this password policy.

- `ds-cfg-allow-expired-password-changes` -- This indicates whether users will be allowed to change their passwords after they have expired. If enabled, this will require the use of the password modify extended operation and the user will need to issue that request anonymously and include the current password in that request.

- `ds-cfg-allow-multiple-password-values` -- This indicates whether the user will be allowed to have multiple distinct password values. This has historically been possible because the `userPassword` attribute is multivalued, but it is strongly discouraged because it is difficult or impossible to ensure strict compliance to password policies if users have multiple different passwords.

- `ds-cfg-allow-pre-encoded-passwords` -- This indicates whether users will be allowed to pre-encode their passwords when changing them. This has historically been allowed, but it is discouraged because if users are allowed to pre-encode their passwords then it is not possible for the server to determine whether it would pass the requirements set by password validators (e.g., whether the given password is too short or is in the password history).

- `ds-cfg-allow-user-password-changes` -- This indicates whether users will be allowed to change their own passwords.

- `ds-cfg-default-password-storage-scheme` -- This specifies the default password storage scheme that will be used to encode new passwords. This is actually a multivalued attribute, and if multiple values are provided then new passwords will be encoded using all of those schemes. This could be beneficial if there are applications that expect passwords to be encoded in a particular scheme.

- `ds-cfg-deprecated-password-storage-scheme` -- This specifies one or more password storage schemes that are considered deprecated. Any user password encoded using a deprecated storage scheme will automatically be re-encoded using the default scheme(s) the next time that user successfully authenticates to the server.

- `ds-cfg-expire-passwords-without-warning` -- This indicates whether user passwords will be allowed to expire even if the user has not yet seen a password expiration warning. If this is set to "false", then even if the password expiration time has passed the user will always be guaranteed to see at least one warning message, and the expiration time will be reset to the current time plus the warning interval.

- `ds-cfg-force-change-on-add` -- This indicates whether users will be required to change their passwords the first time they use their accounts before they will be allowed to perform any other operation.

- `ds-cfg-force-change-on-reset` -- This indicates whether users will be required to change their passwords after an administrative password reset before they will be allowed to perform any other operation.

- `ds-cfg-grace-login-count` -- This specifies the maximum number of grace logins that a user should be given. A grace login makes it possible for a users to authenticate to the server even after the password has expired, but they will not be allowed to do anything else until they have changed their password.

- `ds-cfg-idle-lockout-interval` -- This specifies the maximum length of time that a user account may remain idle (i.e., that the user may go without authenticating to the directory) before the server will lock the account. This will only be enforced if last login time tracking is enabled, and if the idle lockout interval is set to a nonzero value. The value of this configuration attribute should be an integer followed by a space and one of the following units:
  - seconds
  - minutes
  - hours
  - days
  - weeks

- `ds-cfg-last-login-time-attribute` -- This specifies the name of the attribute in the user's entry that will be used to hold the last login time for that user. If this is provided, then the specified attribute must either be defined as an operational attribute in the server schema, or it must be allowed by at least one of the objectclasses in the user's entry.

- `ds-cfg-last-login-time-format` -- This specifies the format string that should be used to generate the last login time values. This may be any valid format string that can be used in conjunction with the `java.text.SimpleDateFormat` class.

- `ds-cfg-lockout-duration` -- This specifies the length of time that a user account should remain locked due to failed authentication attempts before it is automatically unlocked. A value of "0 seconds" indicates that locked accounts will not be automatically unlocked and must be reset by an administrator. The value of this configuration attribute must be an integer followed by a space and one of the following units:
  - seconds
  - minutes
  - hours
  - days
  - weeks

- `ds-cfg-lockout-failure-count` -- This specifies the number of authentication failures required to lock a user account, either temporarily or permanently. A value of zero indicates that automatic lockout will not be enabled.
-
- `ds-cfg-lockout-failure-expiration-interval` -- This specifies the maximum length of time that a previous failed authentication attempt should be counted toward a lockout failure (the record of all previous failed attempts will always be cleared upon a successful authentication). A value of "0 seconds" indicates that failed attempts will never be automatically expired. The value of this configuration attribute must be an integer followed by a space and one of the following units:
  - seconds
  - minutes
  - hours
  - days
  - weeks

- `ds-cfg-maximum-password-age` -- This specifies the maximum length of time that a user will be allowed to keep the same password before choosing a new one. This is also often known as the password expiration interval. A value of "0 seconds" indicates that passwords will never expire. The value of this configuration attribute must be an integer followed by a space and one of the following units:
  - seconds
  - minutes
  - hours
  - days
  - weeks

- `ds-cfg-maximum-password-reset-age` -- This specifies the maximum length of time that users will be allowed to change their passwords after they have been administratively reset before they are locked. This is only applicable if the `ds-cfg-force-change-on-reset` attribute is set to "true". A value of "0 seconds" indicates that there will not be any limit on the length of time that users have to change their passwords after administrative resets. The value of this configuration attribute must be an integer followed by a space and one of the following units:
  - seconds
  - minutes
  - hours
  - days
  - weeks

- `ds-cfg-minimum-password-age` -- This specifies the minimum length of time that a user will be required to have a password value before it can be changed again. Providing a nonzero value will help ensure that users are not allowed to repeatedly change their passwords in order to flush their previous password from the history so it can be reused. The value of this configuration attribute must be an integer followed by a space and one of the following units:
  - seconds
  - minutes
  - hours
  - days
  - weeks

- `ds-cfg-password-attribute` -- This specifies the attribute in the user's entry that will hold the encoded password(s) for the user. The specified attribute must be defined in the server schema, and it must have either the user password syntax or the authentication password syntax.

- `ds-cfg-password-change-requires-current-password` -- This indicates whether users will be required to provide their current password when setting a new password. If this is "true", then users will be required to use the password change extended operation, and they must provide their current password in that request.

- `ds-cfg-password-expiration-warning-interval` -- This specifies the length of time before the password expires that the users should start to receive notification that it is about to expire. This must be given a nonzero value if the `ds-cfg-expire-passwords-without-warning` attribute is set to "false". The value of this configuration attribute must be an integer followed by a space and one of the following units:
  - seconds
  - minutes
  - hours
  - days
  - weeks

- `ds-cfg-password-generator-dn` -- This specifies the DN of the configuration entry for the password generator that should be used in conjunction with this password policy. The password generator will be used in conjunction with the password modify extended operation to provide a new password for cases in which the client did not include one in the request. If no password generator DN is specified, then the password modify extended operation will not automatically generate passwords for users.

- `ds-cfg-password-validator-dn` -- This specifies the DNs of the configuration entries for password validators that should be used in conjunction with this password policy. The password validators will be invoked whenever a user attempts to provide a new password in order to determine whether that new password is acceptable.

- `ds-cfg-previous-last-login-time-format` -- This specifies the format string that might have been used in the past for older last login time values. This will not be necessary unless the last login time feature is enabled and the format in which the values are stored has been changed.

- `ds-cfg-require-change-by-time` -- This specifies a timestamp (in UTC form) that will be used to define a time by which all users with this password policy will be required to change their passwords. This option works independently of password expiration (i.e., so it is possible to force all users to change their passwords at some point even if password expiration is disabled).

- `ds-cfg-require-secure-authentication` -- This indicates whether users with this password policy will be required to authenticate in a secure manner (i.e., either using a secure communication mechanism like SSL, or a secure SASL mechanism like DIGEST-MD5, EXTERNAL, or GSSAPI that does not expose the password in the clear).

- `ds-cfg-require-secure-password-changes` -- This indicates whether users with this password policy will be required to make password changes in a secure manner (e.g., over a secure communication channel like SSL).

- `ds-cfg-skip-validation-for-administrators` -- This indicates whether the new password provided during an administrative password reset will be subject to validation (i.e., the new password must be accepted by all of the password validators) just like it is for user password changes.

The Password Storage Scheme Configuration

The "cn=Password Storage Schemes,cn=config" configuration entry is the parent entry for all password storage schemes defined in the OpenDS Directory Server. Password storage schemes have two primary roles in the server:

- To encode new passwords provided by users so that they are stored in an encoded manner. This will generally make it difficult or impossible for someone to determine the clear-text passwords from the encoded values.

- To determine whether a clear-text password provided by a client matches the encoded value stored in the server.

All password storage scheme configuration entries must contain the `ds-cfg-password-storage-scheme` structural objectclass, which defines the following configuration attributes:

- `ds-cfg-password-storage-scheme-class` -- Specifies the fully-qualified name of the Java class that provides the storage scheme implementation. This class must be a concrete subclass of the `org.opends.server.api.PasswordStorageScheme` superclass.

- `ds-cfg-password-storage-scheme-enabled` -- This indicates whether the password storage scheme is enabled for use.

There are multiple password storage schemes defined in the server, but none of them require any additional configuration.

The Password Validator Configuration

The "cn=Password Validators,cn=config" configuration entry is the parent for all password validators defined in the OpenDS Directory Server. Password validators are responsible for determining whether a proposed password is acceptable for use, and could include checks like ensuring it meets minimum length requirements, that it has an appropriate range of characters, that it is not in the history, etc. The password policy for a user specifies the set of password validators that should be used whenever that user provides a new password.

All password validator configuration entries must contain the `ds-cfg-password-validator` structural objectclass, which defines the following configuration attributes:

- `ds-cfg-password-validator-class` -- This specifies the fully-qualified name of the Java class that provides the password validator logic. This must be a concrete subclass of the `org.opends.server.api.PasswordValidator` superclass.

- `ds-cfg-password-validator-enabled` -- This indicates whether the password validator is enabled for use in the server.

There can be multiple types of password validators defined in the OpenDS Directory Server, and they will be described individually in the remainder of this section.

The Length-Based Password Validator

The length-based password validator may be used to ensure that user passwords meet a given set of minimum and/or maximum length requirements in terms of the number of characters that they contain. The logic for this password validator is implemented in the `org.opends.server.extensions.LengthBasedPasswordValidator` class, and the configuration entry should contain the `ds-cfg-length-based-password-validator` objectclass that includes the following configuration attributes:

- `ds-cfg-maximum-password-length` -- This specifies the maximum number of characters that will be allowed for the password. A value of zero indicates that there is no maximum length.

- `ds-cfg-minimum-password-length` -- This specifies the minimum number of characters that will be allowed for the password. A value of zero indicates that there is no minimum length.

The Plugin Configuration

The "cn=Plugins,cn=config" configuration entry is the parent for all plugins defined in the OpenDS Directory Server. Plugins are responsible for injecting custom logic into the course of processing an operation or other well-defined points within the server. There are several different kinds of plugins, including:

- Pre-parse plugins may be used to alter the contents of a request before the server begins to process it.
- Pre-operation plugins may be used to perform any processing that may be required immediately before the server performs the core action associated with that operation (e.g., before an updated entry is stored in the backend for a modify operation).
- Post-operation plugins may be used to perform any processing that may be required immediately after the server performs the core action associated with that operation but before the response is sent to the client.
- Post-response plugins may be used to perform any processing that may be required for an operation that does not need to occur before the response is sent to the client.
- Search result entry plugins may be used to perform any processing that may be required immediately before the server sends a search result entry to a client (including altering the contents of the entry or preventing it from being sent).
- Search result reference plugins may be used to perform any processing that may be required immediately before the server sends a search reference (i.e., a referral) to a client (including altering the contents of the referral or preventing it from being sent).
- Intermediate response plugins may be used to perform any processing that may be required immediately before the server sends an intermediate response to a client (including altering the contents of the intermediate response or preventing it from being sent).
- Post-connect plugins may be used to perform any processing that may be required immediately after the server accepts a new connection from a client (including terminating that connection).
- Post-disconnect plugins may be used to perform any processing that may be required immediately after a client connection is closed for some reason (regardless of whether the closure was initiated by the client or the server).
- Startup plugins may be used to perform any processing that may be required when the server is in the process of being started.

- Shutdown plugins may be used to perform any processing that may be required when the server is in the process of performing a graceful shutdown.

- LDIF import plugins may be used to perform any processing that may be required for any entries read during the LDIF import process (including altering the contents of the entry or preventing it from being imported).

- LDIF export plugins may be used to perform any processing that may be required for any entries to be written during the LDIF export process (including altering the contents of the entry or preventing it from being exported).

All plugin configuration entries must contain the `ds-cfg-plugin` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-plugin-class` -- This specifies the fully-qualified name of the Java class that provides the plugin logic. All plugins in the OpenDS Directory Server must be concrete subclasses of the `org.opends.server.api.plugin.DirectoryServerPlugin` superclass.

- `ds-cfg-plugin-enabled` -- This indicates whether the plugin will be enabled for use in the server.

- `ds-cfg-plugin-type` -- This specifies the plugin type(s) for the plugin. Valid plugin types include:
  - preParseAbandon
  - preParseAdd
  - preParseBind
  - preParseCompare
  - preParseDelete
  - preParseExtended
  - preParseModify
  - preParseModifyDN
  - preParseSearch
  - preParseUnbind
  - preOperationAdd
  - preOpertionBind
  - preOperationCompare
  - preOperationDelete
  - preOperationExtended
  - preOperationModify
  - preOperationModifyDN
  - preOperationSearch
  - postOperationAdd
  - postOperationBind
  - postOperationCompare
  - postOperationDelete

- postOperationExtended
- postOperationModify
- postOperationModifyDN
- postOperationSearch
- postResponseAbandon
- postResponseAdd
- postResponseBind
- postResponseCompare
- postResponseDelete
- postResponseExtended
- postResponseModify
- postResponseModifyDN
- postResponseSearch
- postResponseUnbind
- searchResultEntry
- searchResultReference
- intermediateResponse
- startup
- shutdown
- ldifImport
- ldifExport

There can be multiple types of plugins defined in the OpenDS Directory Server, and they will be described individually in the remainder of this section.

The Entry UUID Plugin

The entry UUID plugin is responsible for ensuring that all entries added to the server, whether through an LDAP add operation or via an LDIF import, are assigned an `entryUUID` operational attribute if they don't already have one. The `entryUUID` attribute contains a universally unique identifier that can be used to identify an entry in a manner that will not change (even in the event of a modify DN operation). This plugin generates a random UUID for entries created by an add operation, but the UUID is constructed from the DN of the entry during an LDIF import (which means that the same LDIF file can be imported on different systems but will still get the same value for the `entryUUID` attribute). This behavior is based on the specification contained in RFC 4530.

The implementation for the entry UUID plugin is contained in the `org.opends.server.plugins.EntryUUIDPlugin` class. It must be configured with the preOperationAdd and ldifImport plugin types, but it does not have any other custom configuration.

The LastMod Plugin

The LastMod plugin is responsible for ensuring that the `creatorsName` and `createTimestamp` attributes are included in the user's entry when it is added to the server, and that the `modifiersName` and `modifyTimestamp` attributes are included for modify and modify DN operations. This behavior is described in RFC 4512.

The implementation for the LastMod plugin is contained in the `org.opends.server.plugins.LastModPlugin` class. It must be configured with the preOperationAdd, preOperationModify, and preOperationModifyDN plugin types, but it does not have any other custom configuration.

The LDAP Attribute Description List Plugin

The LDAP attribute description list plugin provides the ability for clients to include an attribute list in a search request that names objectclasses instead of (or in addition to) attributes. For example, if a client wishes to retrieve all of the attributes in the `inetOrgPerson` objectclass, then it is possible for that client to simply include "@inetOrgPerson" in the attribute list rather than naming all of those attributes individually. This behavior is based on the specification contained in RFC 4529.

The implementation for the LDAP attribute description list plugin is contained in the `org.opends.server.plugins.LDAPADListPlugin` class. It must be configured with the preParseSearch plugin type, but does not have any other custom configuration.

The Password Policy Import Plugin

The password policy import plugin is responsible for ensuring that all clear-text user passwords are encoded using the proper password storage schemes during an LDIF import.

The implementation for this plugin is contained in the `org.opends.server.plugins.PasswordPolicyImportPlugin` class. It must be configured with the ldifImport plugin type, but it does not have any other custom configuration.

The Profiler Plugin

The profiler plugin provides a basic in-server profiling capability that can be used to monitor performance characteristics and identify potential hotspots in the OpenDS Directory Server. It is based on the relatively simple principle of periodically capturing stack traces from all threads in the JVM and collecting them so that they can be aggregated and correlated for later analysis. This can be a very useful tool for debugging performance problems without the need for any other external tools and without the need to restart the server to start and stop profiling. The plugin has absolutely no impact on performance while it is not in use, and can be relatively inexpensive even when it is capturing data.

The implementation for this plugin is contained in the `org.opends.server.plugins.profiler.ProfilerPlugin` class. This plugin must be configured with the startup plugin type, but has a number of other configurable attributes as allowed by the `ds-cfg-profiler-plugin` structural objectclass:

- `ds-cfg-enable-profiling-on-startup` -- This indicates whether the profiler should be automatically enabled when it is loaded so that it can capture information about parts of the OpenDS Directory Server startup process. This should be set to "false" unless startup profiling is required because otherwise the volume of data that will be collected can cause the server to run out of memory if it is not turned off in a timely manner.

- `ds-cfg-profile-directory` -- This specifies the path to the filesystem directory into which the profile output data should be written. This may be either an absolute path or a path that is relative to the root of the OpenDS Directory Server instance.

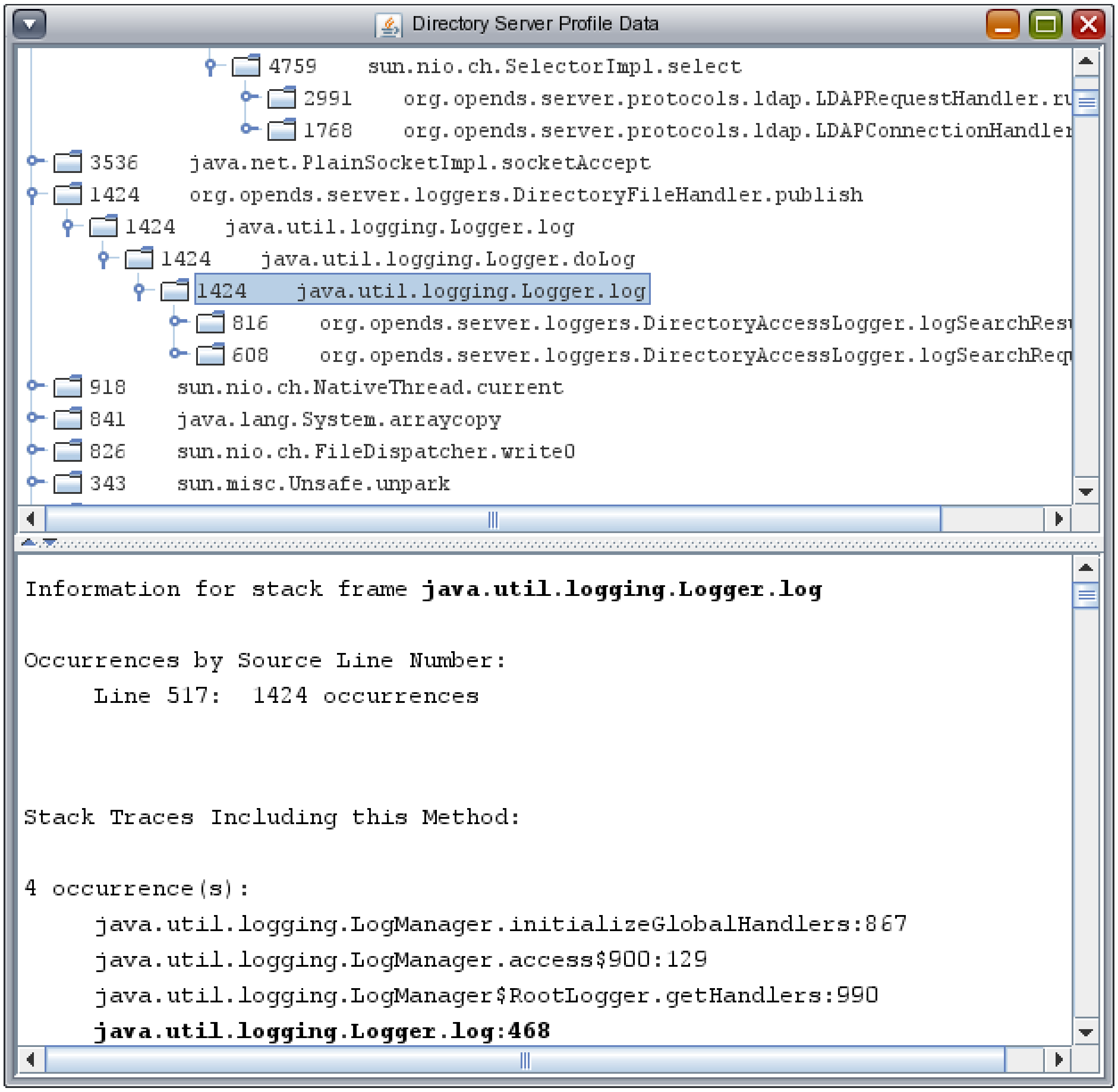
- `ds-cfg-profile-sample-interval` -- This specifies the sample interval in milliseconds that should be used. When capturing data, the profiler thread will sleep for this length of time between calls to obtain traces for all threads running in the JVM.

- `ds-cfg-profile-action` -- This specifies the action that the profiler plugin should take. It should be modified via LDAP or JMX in order to perform the desired action. It may be given one of the following values:
  - start -- Causes the profiler plugin to start capturing data.
  - stop -- Causes the profiler plugin to stop capturing data and write the results to a file in the profile directory.
  - cancel -- Causes the profiler plugin to stop capturing data and throw away any results that had been captured.

The data file that the profiler collects is written in a binary form, and it is necessary to use a tool provided with the server in order to be able to view its contents. It may be invoked as a purely command-line tool, but it can also provide a graphical interface to examine the data. In order to run the tool, go into the OpenDS instance root and issue the following command (all on one line):

```
java -cp lib/OpenDS.jar
org.opends.server.plugins.profiler.ProfileViewer -f {dataFile} -g
```

where *{dataFile}* is the path to the data file containing the data that was collected by the profiler plugin. This will cause the profiler GUI interface to be displayed, which should look like the following:



```
            4759    sun.nio.ch.SelectorImpl.select
              2991    org.opends.server.protocols.ldap.LDAPRequestHandler.ru
              1768    org.opends.server.protocols.ldap.LDAPConnectionHandler
3536    java.net.PlainSocketImpl.socketAccept
1424    org.opends.server.loggers.DirectoryFileHandler.publish
  1424    java.util.logging.Logger.log
    1424    java.util.logging.Logger.doLog
      1424    java.util.logging.Logger.log
        816     org.opends.server.loggers.DirectoryAccessLogger.logSearchResu
        608     org.opends.server.loggers.DirectoryAccessLogger.logSearchReq
918     sun.nio.ch.NativeThread.current
841     java.lang.System.arraycopy
826     sun.nio.ch.FileDispatcher.write0
343     sun.misc.Unsafe.unpark
```

```
Information for stack frame java.util.logging.Logger.log

Occurrences by Source Line Number:
     Line 517:  1424 occurrences



Stack Traces Including this Method:

4 occurrence(s):
     java.util.logging.LogManager.initializeGlobalHandlers:867
     java.util.logging.LogManager.access$900:129
     java.util.logging.LogManager$RootLogger.getHandlers:990
     java.util.logging.Logger.log:468
```

The top panel of the data displayed will include a list of all the leaf methods seen by the profiler, ordered by the number of profile intervals in which that method was seen. Each node may be expanded to show the method(s) that called it and number of times they appeared in that particular stack. The bottom panel will display a list of all stack traces in which the highlighted method appears.

To invoke the profiler data viewer in command-line mode (to create text-only output), simply remove the "`-g`" option from the command line.

The Root DN Configuration

The "cn=Root DNs,cn=config" configuration entry is the parent for all root DNs (also called root users) defined in the OpenDS Directory Server. Root users in OpenDS are very similar to root users in traditional UNIX systems in that they have the ability to perform just about any kind of operation. At the present time, the access control subsystem is not yet in place and therefore there are no advantages given to root users over anyone else, but in the future the primary benefit extended to root users will be the ability to bypass access control checking for any entries in the server. It is expected that a privileges subsystem may also provide the ability for non-root users to bypass access controls for some or all parts of the server.

All root DN entries must contain the `ds-cfg-root-dn` auxiliary objectclass, which includes the following configuration attribute:

- `ds-cfg-alternate-bind-dn` -- This defines one or more alternate DNs that may be supplied when binding to the server as this user. This is provided primarily as a convenience for legacy applications that might expect a particular type of root DN (e.g., "cn=Directory Manager").

Because the `ds-cfg-root-dn` objectclass is an auxiliary class, each root DN entry must have its own structural class (e.g., `inetOrgPerson`) that allows the additional attributes to include in that entry. Because root users in the OpenDS Directory Server have actual entries (as opposed to the `nsslapd-rootdn` and `nsslapd-rootpw` attributes of the Sun Java System Directory Server), it is possible to provide other attributes that might be necessary to allow for stronger authentication mechanisms (e.g., a certificate for use with SASL EXTERNAL, or information used for identity mapping). It is also possible to have multiple root DNs in the OpenDS Directory Server, which means that each of them can have their own password (and associated password policy) and other settings on an individual basis rather than requiring all administrators to share a single root DN and password.

The Root DSE Configuration

The "cn=Root DSE,cn=config" entry contains the configuration information for the root DSE in the OpenDS Directory Server. The root DSE is the entry with the null DN (i.e., a DN of "", containing zero RDN components), and it exists to provide useful information about the capabilities of the server, including about the SASL mechanisms, controls, extended operations, and other types of features that the server supports.

The OpenDS Directory Server root DSE implementation does introduce differences in behavior as compared with the Sun Java System Directory Server. The first of these is that most of the attributes contained in this entry are operational attributes and will be treated that way by default, meaning that they will not be returned to the client unless they are explicitly requested. This is the correct behavior as per the LDAP specification, but historically these attributes have always been returned even if not requested despite the fact that they are declared as operational.

Another major change in behavior is that it is now possible to perform onelevel, subtree, and subordinate subtree searches from the root DSE. In the Sun Java System Directory Server, the root DSE was essentially considered a peer to all of the suffixes defined in the server, and performing a onelevel or subtree search below the root DSE did not yield any results. In the OpenDS Directory Server, it is possible to treat the root DSE as a parent to all other suffixes, and have searches below the root DSE performed in all user backends (or using a configurable set of base DNs).

The root DSE configuration entry must contain the `ds-cfg-root-dse` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-show-all-attributes` -- This indicates whether the root DSE should treat all attributes in the root DSE as if they were user attributes, and therefore return them by default if no attributes were requested. This can be used to obtain behavior that is similar to the Sun Java System Directory Server for applications that expect to be able to see these attributes.

- `ds-cfg-subordinate-base-dn` -- This specifies the base DN(s) that should be used for onelevel, subtree, and subordinate subtree searches below the root DSE. By default, all user-defined data suffixes will be used.

The root DSE configuration entry is also special in that any other attributes included in the "cn=Root DSE,cn=config" entry will be directly included added to the root DSE entry itself that is generated upon request. This is similar to the configuration entry for the monitor backend in which any attributes that appear in that entry that aren't involved in the configuration for the monitor backend will be included in the "cn=monitor" entry.

The SASL Mechanism Handler Configuration

The "cn=SASL Mechanisms,cn=config" configuration entry is the parent for all SASL mechanism handlers defined in the OpenDS Directory Server. SASL mechanism handlers are responsible for authenticating users during the course of processing a SASL (Simple Authentication and Security Layer, as defined in RFC 4422) bind.

All SASL mechanism handler configuration entries must contain the `ds-cfg-sasl-mechanism-handler` structural objectclass, which defines the following configuration attributes:

- `ds-cfg-sasl-mechanism-handler-class` -- This specifies the fully-qualified name of the Java class that provides the SASL mechanism handler logic. This must be a concrete subclass of the `org.opends.server.api.SASLMechanismHandler` superclass.

- `ds-cfg-sasl-mechanism-handler-enabled` -- This indicates whether the SASL mechanism handler is enabled for use in the OpenDS Directory Server.

There can be multiple types of SASL mechanism handlers defined in the OpenDS Directory Server, and they will be described individually in the remainder of this section.

The ANONYMOUS SASL Mechanism Handler

The ANONYMOUS SASL mechanism provides the ability for clients to perform an anonymous bind using a SASL mechanism. The only real benefit that this provides over a normal anonymous bind (i.e., using simple authentication with no password) is that the ANONYMOUS SASL mechanism also allows the client to include a trace string in the request, which could help identify the application that performed the bind (although since there is no authentication, there's no assurance that some other client didn't spoof that trace string).

The ANONYMOUS SASL mechanism handler is implemented in the `org.opends.server.extensions.AnonymousSASLMechanismHandler` class. It does not require any additional configuration.

The CRAM-MD5 SASL Mechanism Handler

The CRAM-MD5 SASL mechanism provides the ability for clients to perform password-based authentication in a manner that does not expose their password in the clear. Rather than including the password in the bind request, the CRAM-MD5 mechanism uses a two-step process in which the client simply needs to prove that it knows the password. The server will send randomly-generated data to the client that is to be used in the process, which makes it very resistant to replay attacks, and the one-way message digest algorithm ensures that the original clear-text password is not exposed.

Note that the algorithm used by the CRAM-MD5 mechanism requires that both the client and the server have access to the clear-text password (or potentially a value that is trivially-derived from the clear-text password). In order to authenticate to the server using CRAM-MD5, the password for a user's account must be encoded using a reversible password storage scheme that allows the server to have access to the clear-text value.

The CRAM-MD5 SASL mechanism handler is implemented in the `org.opends.server.extensions.CRAMMD5SASLMechanismHandler` class. The configuration entry must include the `ds-cfg-cram-d5-sasl-mechanism-handler` structural objectclass, which includes the following attribute:

- `ds-cfg-identity-mapper-dn` -- This specifies the DN of the configuration entry for the identity mapper that should be used with this SASL mechanism handler. It will be used to map the authentication ID included in the SASL bind request to the corresponding user in the directory.

The DIGEST-MD5 SASL Mechanism Handler

The DIGEST-MD5 SASL mechanism is very similar to the CRAM-MD5 mechanism in that it allows for password-based authentication without exposing the password in the clear (although it does require that both the client and the server have access to the clear-text password). Like the CRAM-MD5 mechanism, it uses data that is randomly generated by the server to make it resistant to replay attacks, but it also includes randomly-generated data from the client, which makes it also resistant to problems resulting from weak server-side random number generation.

The DIGEST-MD5 SASL mechanism handler is implemented in the `org.opends.server.extensions.DIGESTMD5SASLMechanismHandler` class. The configuration entry must include the `ds-cfg-digest-md5-sasl-mechanism-handler` structural objectclass, which includes the following attribute:

- `ds-cfg-identity-mapper-dn` -- This specifies the DN of the configuration entry for the identity mapper that should be used with this SASL mechanism handler. It will be used to map the authentication ID and/or authorization ID included in the SASL bind request to the corresponding user in the directory.

- `ds-cfg-realm` -- This specifies the realm that should be used for the DIGEST-MD5 authentication.

The EXTERNAL SASL Mechanism Handler

The EXTERNAL SASL mechanism provides the ability for clients to authenticate themselves to the server using identification information that is provided outside of the LDAP protocol. At present, this identification may be performed using an SSL client certificate. When the client performs SSL negotiation with the server, either by establishing an SSL-based connection or through the use of the StartTLS extended operation, if the client sends its own certificate to the server then that certificate may be used for authentication via SASL EXTERNAL.

The EXTERNAL SASL mechanism handler is implemented in the `org.opends.server.extensions.ExternalSASLMechanismHandler` class. The configuration entry must include the `ds-cfg-external-sasl-mechanism-handler` structural objectclass, which includes the following attributes:

- `ds-cfg-certificate-attribute` -- This specifies the attribute in the user's entry that will hold the public portion of the client's certificate. If the server is configured to verify the certificate presented by the client, then this attribute will be used to obtain the certificate from the user's entry.

- `ds-cfg-client-certificate-validation-policy` -- This indicates whether the server will attempt to verify that the certificate presented by the client during SSL negotiation matches the certificate stored in the corresponding user entry. This can help provide additional assurance of the client's identity. The value of this configuration attribute must be one of the following strings:
  - always -- This indicates that the server will always attempt to compare the certificate provided by the client with a certificate from the associated user's entry. If the user's entry does not contain any certificates, or if none of those certificates match the one provided by the client, then the authentication will fail.
  - never -- This indicates that the server will never attempt to compare the certificate provided by the client with a certificate from the associated user's entry.
  - ifpresent -- This indicates that the server will attempt to compare the certificate provided by the client with a certificate from the associated user's entry. If the user's entry does not contain any certificates, then the authentication will be allowed to continue. If the user's entry does contain one or more certificates but none of them match the certificate provided by the client, then the authentication will fail.

Note that the process of actually mapping the client certificate to a user account in the directory is performed using the SSL certificate mapper. At the present time, SSL settings are shared throughout the server, but this is expected to be revised in the future so that individual components can use their own settings for SSL-based communication. At that time, the `ds-cfg-external-sasl-mechanism-handler` objectclass will also be updated to include additional information about which certificate mapper should be used to identify the user.

The GSSAPI SASL Mechanism Handler

The GSSAPI SASL mechanism provides the ability for clients to authenticate themselves to the server using existing authentication in a Kerberos environment. This provides the ability to achieve single sign-on for Kerberos-based clients.

The GSSAPI SASL mechanism handler is implemented in the `org.opends.server.extensions.GSSAPISSASLMechanismHandler` class. The configuration entry must contain the `ds-cfg-gssapi-sasl-mechanism-handler` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-identity-mapper-dn` -- This specifies the DN of the configuration entry for the identity mapper that should be used with this SASL mechanism handler. It will be used to map the authentication ID and/or authorization ID included in the SASL bind request to the corresponding user in the directory.

- `ds-cfg-kdc-address` -- This specifies the address of the KDC that will be used for GSSAPI authentication. If this is not provided, then the server will attempt to determine it from the system-wide Kerberos configuration.

- `ds-cfg-keytab` -- This specifies the path to the keytab file that contains the secret key for the Kerberos principal that should be used when processing GSSAPI authentication. If this is not provided, then the system-wide default keytab file will be used.

- `ds-cfg-realm` -- This specifies the realm that should be used for GSSAPI authentication. If this is not provided, then the server will attempt to determine it from the system-wide Kerberos configuration.

- `ds-cfg-server-fqdn` -- This specifies the fully-qualified domain name that should be used when processing GSSAPI authentication. If this is not provided, then the server will attempt to determine it from the system-wide Kerberos configuration.

The PLAIN SASL Mechanism Handler

The PLAIN SASL mechanism provides the ability for clients to authenticate using a username and password. This is very similar to standard LDAP simple authentication, with the exception that it can authenticate based on an authentication ID (e.g., a username) rather than requiring a full DN, and it can also include an authorization ID in addition to the authentication ID. Note that the SASL PLAIN mechanism does not make any attempt to protect the password.

The PLAIN SASL mechanism handler is implemented in the `org.opends.server.extensions.PlainSASLMechanismHandler` class. The configuration entry must contain the `ds-cfg-plain-sasl-mechanism-handler` structural objectclass, which includes the following configuration attribute:

- `ds-cfg-identity-mapper-dn` -- This specifies the DN of the configuration entry for the identity mapper that should be used with this SASL mechanism handler. It will be used to map the authentication ID and/or authorization ID included in the SASL bind request to the corresponding user in the directory.

The SSL-Related Configuration

SSL (and its successor, TLS) is one of the most popular ways of securing network communication. It uses asymmetric encryption (also called public key encryption) for the benefits that it provides in terms of key management, and it uses symmetric encryption (also called bulk encryption) for the benefits that it provides in performance.

There are three primary components to the SSL subsystem of the OpenDS Directory Server:

- Certificate mappers are used to correlate certificates presented by an SSL client to the appropriate users in the directory data.

- Key manager providers are used to provide access to private keys, which are always needed when acting as an SSL server, and may be needed when acting as an SSL client if authentication is involved.

- Trust manager providers are used to determine whether to trust a certificate presented by a client or another server.

The OpenDS Directory Server implementation is relatively flexible, in that it makes each of these components modular. For example, in order to change the way in which the server obtains the private key it uses to accept SSL communication, it is only necessary to change the key manager provider to an appropriate alternate implementation. However, at the present time it does fall somewhat short because it assumes that there will be only a single SSL configuration for the entire server. This is not necessarily the case, and it may be beneficial for different components to be able to have separate configurations. In the future, the OpenDS Directory Server configuration will be altered so that this is possible. The information in the remainder of this section describes the current implementation.

The Certificate Mapper Configuration

The certificate mapper is responsible for identifying a directory user associated with a given client certificate. This is used by the SASL EXTERNAL mechanism handler in order to authenticate users. The configuration for the certificate mapper is contained in the cn=Certificate Mapper,cn=SSL,cn=config configuration entry, which must have the `ds-cfg-certificate-mapper` structural objectclass. This objectclass includes the following configuration attributes:

- `ds-cfg-certificate-mapper-class` -- This specifies the fully-qualified name of the Java class that provides the certificate mapper implementation. This must be a concrete subclass of the `org.opends.server.api.CertificateMapper` superclass.

- `ds-cfg-certificate-mapper-enabled` -- This indicates whether the certificate mapper should be enabled for use in the OpenDS Directory Server. If it is not enabled, then SASL EXTERNAL authentication will not be possible.

At present, there is a single class provided with the OpenDS Directory Server that provides a certificate mapper implementation. That is the `org.opends.server.extensions.SubjectEqualsDNCertificateMapper` class, and as its name implies it assumes that the subject of the SSL client certificate is equal to the DN of the corresponding user entry. This class does not require any additional configuration.

The Key Manager Provider Configuration

The key manager provider is responsible for obtaining a `javax.net.ssl.KeyManager` instance that will be used to obtain private key information. The configuration for the key manager provider is contained in the cn=Key Manager Provider,cn=SSL,cn=config entry, which must have the `ds-cfg-key-manager-provider` structural objectclass. This objectclass includes the following configuration attributes:

- `ds-cfg-key-manager-provider-class` -- This specifies the fully-qualified name of the Java class that contains the key manager provider implementation. This must be a concrete subclass of the `org.opends.server.api.KeyManagerProvider` superclass.

- `ds-cfg-key-manager-provider-enabled` -- This indicates whether the key manager provider should be enabled for use in the OpenDS Directory Server.

The File-Based Key Manager Provider

The file-based key manager provider accesses key information in a file on the local filesystem. Multiple file formats may be supported, depending on the providers supported by the underlying Java runtime.

The file-based key manager provider implementation is contained in the `org.opends.server.extensions.FileBasedKeyManagerProvider` class. It will make use of the following configuration attributes contained in the `ds-cfg-file-based-key-manager-provider` structural objectclass:

- `ds-cfg-key-store-file` -- This specifies the path to the file containing the private key information. It may be an absolute path, or a path that is relative to the OpenDS Directory Server instance root.

- `ds-cfg-key-store-type` -- This specifies the format for the data in the key store file. This may vary based on the underlying runtime environment, but the following types will generally be available:
  - JKS -- This is the default Java Key Store format.

    - PKCS12 -- This is the standard PKCS#12 format.

- `ds-cfg-key-store-pin` -- This specifies the PIN to use to access the contents of the key store.

- `ds-cfg-key-store-pin-environment-variable` -- This specifies the name of an environment variable that contains the PIN to use to access the contents of the key store.

- `ds-cfg-key-store-pin-file` -- This specifies the path to a file containing the PIN to use to access the contents of the key store. It may be an absolute path, or a path that is relative to the OpenDS Directory Server instance root.

- `ds-cfg-key-store-pin-property` -- This specifies the name of a Java property that contains the PIN to use to access the contents of the key store.

Exactly one mechanism for obtaining the PIN should be provided. Regardless of its location, the PIN should not be obscured in any way.

The PKCS#11 Key Manager Provider

The PKCS#11 key manager provider provides the ability for the server to access the private key information through the PKCS#11 interface. This standard interface is used by cryptographic accelerators and hardware security modules.

The PKCS#11 key manager provider implementation is contained in the `org.opends.server.extensions.PKCS11KeyManagerProvider` class. It will make use of the following configuration attributes included in the `ds-cfg-pkcs11-key-manager-provider` structural objectclass:

- `ds-cfg-key-store-pin` -- This specifies the PIN to use to access the contents of the key store.

- `ds-cfg-key-store-pin-environment-variable` -- This specifies the name of an environment variable that contains the PIN to use to access the contents of the key store.

- `ds-cfg-key-store-pin-file` -- This specifies the path to a file containing the PIN to use to access the contents of the key store. It may be an absolute path, or a path that is relative to the OpenDS Directory Server instance root.

- `ds-cfg-key-store-pin-property` -- This specifies the name of a Java property that contains the PIN to use to access the contents of the key store.

Exactly one mechanism for obtaining the PIN should be provided. Regardless of its location, the PIN should not be obscured in any way.

The Trust Manager Provider Configuration

The trust manager provider is responsible for obtaining a `javax.net.ssl.TrustManager` instance that will be used to determine whether to trust any certificates that are presented to the server. The configuration for the trust manager provider is contained in the cn=Trust Manager Provider,cn=SSL,cn=config entry, which must have the `ds-cfg-trust-manager-provider` structural objectclass. This objectclass includes the following configuration attributes:

- `ds-cfg-trust-manager-provider-class` -- This specifies the fully-qualified name of the Java class that contains the trust manager provider implementation. This must be a concrete subclass of the `org.opends.server.api.TrustManagerProvider` superclass.

- `ds-cfg-trust-manager-provider-enabled` -- This indicates whether the trust manager provider should be enabled for use in the OpenDS Directory Server.

The File-Based Trust Manager Provider

The file-based trust manager provider accesses trust information in a file on the local filesystem. Multiple file formats may be supported, depending on the providers supported by the underlying Java runtime.

The file-based trust manager provider implementation is contained in the `org.opends.server.extensions.FileBasedTrustManagerProvider` class. It will make use of the following configuration attributes contained in the `ds-cfg-file-based-trust-manager-provider` structural objectclass:

- `ds-cfg-trust-store-file` -- This specifies the path to the file containing the trust information. It may be an absolute path, or a path that is relative to the OpenDS Directory Server instance root.

- `ds-cfg-trust-store-type` -- This specifies the format for the data in the trust store file. This may vary based on the underlying runtime environment, but the following types will generally be available:
  - JKS -- This is the default Java Key Store format.
  - PKCS12 -- This is the standard PKCS#12 format.

- `ds-cfg-trust-store-pin` -- This specifies the PIN to use to access the contents of the trust store.

- `ds-cfg-trust-store-pin-environment-variable` -- This specifies the name of an environment variable that contains the PIN to use to access the contents of the trust store.

- `ds-cfg-trust-store-pin-file` -- This specifies the path to a file containing the PIN to use to access the contents of the trust store. It may be an absolute path, or a path that is relative to the OpenDS Directory Server instance root.

- `ds-cfg-trust-store-pin-property` -- This specifies the name of a Java property that contains the PIN to use to access the contents of the trust store.

The trust store implementation may not require the use of a PIN in order to access the trust information, since it is not generally considered sensitive. If a PIN is required, exactly one mechanism for obtaining the trust store PIN should be provided. Regardless of its location, the PIN should not be obscured in any way.

The Blind Trust Manager Provider

The blind trust security manager is a very simple implementation that will always trust any certificate presented to it. This may provide some level of convenience, but it can also have significant security problems. In particular, if used in conjunction with the SASL EXTERNAL authentication method, it will malicious users to create their own client certificate to masquerade as any other user. The use of the blind trust manager provider outside of any test environment is strongly discouraged.

The blind trust manager provider implementation is contained in the `org.opends.server.extensions.BlindTrustManagerProvider` class. It does not require any additional configuration.

The Synchronization Provider Configuration

The data replication capabilities of the Sun Java System Directory Server are widely to provide both high availability and horizontal scalability, but it can create something of a management problem. Because of its point-to-point nature, replication agreements must be created between the individual endpoints, and whenever a new server is added into the environment other systems must be updated to know about it. The OpenDS Directory Server, on the other hand, is taking a centralized approach to the problem. One or more changelog servers are created in the environment, and each Directory Server instance is simply pointed at the changelog server(s). This can dramatically simplify the deployment and management of the environment.

There are also a number of enhancements planned for the OpenDS Directory Server in the way that it can communicate with other server instances, and also in the way that other applications can be made aware of changes in the directory environment. For this reason, the term "replication" has been supplanted with "synchronization". All of the replication capabilities of the Sun Java System Directory Server will be retained (although not necessarily implemented in the same manner), and a number of new features will be added. Note that this is currently a work in progress, and much of the expected functionality has not yet been implemented. The documentation contained in this section provides information about the currently-available functionality.

The Multimaster Synchronization Provider Configuration

The multimaster synchronization provider component runs as part of the OpenDS Directory Server and is responsible for communicating with the changelog server. It will send the changelog information about changes made in the local server, and will read information from the changelog about changes made in other servers.

The synchronization provider configuration is actually divided into two entries. The first of these is the "cn=Multimaster Synchronization,cn=Synchronization Providers,cn=config" entry, and it must be present regardless of whether the server is acting as a dedicated changelog, a dedicated data server, or a combined changelog and data server. It must have the ds-cfg-synchronization-provider structural objectclass with the following configuration attributes:

- `ds-cfg-synchronization-provider-class` -- This specifies the fully-qualified name of the Java class that provides the synchronization provider logic. For the multimaster synchronization provider, this should be `org.opends.server.synchronization.MultimasterSynchronization`.

- `ds-cfg-synchronization-provider-enabled` -- This indicates whether the synchronization provider should be enabled for use in the OpenDS Directory Server.

The other component(s) of the multimaster synchronization configuration are the entries that actually configure synchronization for data in the server. These entries are not needed on dedicated changelog servers. They should have the `ds-cfg-synchronization-provider-config` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-changelog-server` -- This specifies the information needed to communicate with the changelog server(s) in the environment. The values of this configuration attribute should contain the address of the changelog server followed by a colon and the port number.

- `ds-cfg-receive-status` -- This indicates whether the server will attempt to receive changes from the changelog. If this is set to "false", then the server will send any changes made locally to the changelog, but will not apply any changes made on any other server. This attribute exists primarily for testing purposes and in general its value should not be altered.

- `ds-cfg-directory-server-id` -- This specifies the unique identifier that has been assigned to this Directory Server instance. All Directory Server instances must have different server IDs.

- `ds-cfg-synchronization-dn` -- This specifies the base DN for which to synchronize changes. In most cases, it should be set to the user data suffix for the Directory Server.

If there are multiple data suffixes in the server, then they will each require separate configuration entries.

The Changelog Configuration

The changelog component of the OpenDS Directory Server is responsible for keeping track of all of the changes in the environment. Each Directory Server instance in a synchronization environment communicates with a changelog, and the individual changelog instances communicate with each other. Unlike the Sun Java System Directory Server, there does not need to be a one-to-one mapping between master Directory Server instances and changelog databases. In fact, it is entirely possible to only run the changelog servers on dedicated systems so that none of the Directory Server instances actually containing the data have a changelog. This can provide a cleaner separation of user data and synchronization metadata, and may also make the synchronization environment more performant and scalable.

The changelog configuration should be defined in the "cn=Changelog Server,cn=Multimaster Synchronization,cn=Synchronization Providers,cn=config" configuration entry. This entry contains the `ds-cfg-synchronization-changelog-server-config` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-changelog-port` -- This specifies the TCP port on which the changelog server will listen for connections from Directory Server instances and other changelog servers.

- `ds-cfg-changelog-server` -- This specifies the address and port to use to communicate with other changelog servers in the environment. Each value for this attribute should consist of the address of a changelog server followed by a colon and the port number on which it is listening.

`ds-cfg-changelog-server-id` -- This specifies the unique identifier that has been assigned to this changelog server instance. All changelog servers must have different server IDs.

The Attribute Syntax Configuration

The "cn=Syntaxes,cn=config" configuration entry is the parent entry for all attribute syntaxes defined in the OpenDS Directory Server. Attribute syntaxes define the types of information that may be stored in the associated attributes.

All attribute syntax configuration entries must contain the `ds-cfg-attribute-syntax` structural objectclass, which includes the following configuration attributes:

- `ds-cfg-syntax-class` -- This specifies the fully-qualified name of the Java class that provides the attribute syntax implementation. This must be a concrete subclass of the `org.opends.server.api.AttributeSyntax` superclass.

- `ds-cfg-syntax-enabled` -- This indicates whether the attribute syntax is enabled for use in the OpenDS Directory Server.

There are a number of attribute syntaxes defined in the OpenDS Directory Server, but none of them require any additional configuration.

The Virtual Attribute Configuration

The "cn=Virtual Attributes,cn=config" configuration entry is the parent for all virtual attribute definitions in the OpenDS Directory Server. The virtual attribute subsystem is not yet implemented in the server.

The Work Queue Configuration

The "cn=Work Queue,cn=config" configuration entry provides the configuration for the server work queue. Whenever the server receives a request from a client, that request is placed in the work queue which is responsible for ensuring that the operation is processed.

The work queue entry must contain the `ds-cfg-work-queue` structural objectclass, which includes the following configuration attribute:

- `ds-cfg-work-queue-class` -- This specifies the fully-qualified name of the Java class that provides the work queue implementation. This must be a concrete subclass of the `org.opends.server.api.WorkQueue` superclass.

Only a single work queue may be defined in the server at any time. Individual work queue implementations will be described in the remainder of this section.

The Traditional Work Queue

The traditional work queue is named that because its implementation is very similar to that used by the Sun Java System Directory Server. It is a FIFO queue serviced by a fixed number of worker threads. However, there are a couple of notable differences in its design:

- The number of worker threads is fixed, but it can be changed on the fly and those changes will take effect immediately. In the Sun Java System Directory Server, changes to the number of worker threads require a server restart to take effect.

- The work queue in the Sun Java System Directory Server is unbounded. If all threads are busy processing existing operations and new requests arrive, they will continue to accumulate in the work queue and the server will appear to be frozen. In the OpenDS Directory Server, it is possible to place a size limit on the work queue so once that many operations are in the queue waiting to be picked up by threads, any new requests received will be rejected with an error message.

The traditional work queue implementation is contained in the `org.opends.server.extensions.TraditionalWorkQueue` class. The following configuration attributes are available as part of the `ds-cfg-traditional-work-queue` objectclass:

- `ds-cfg-max-work-queue-capacity` -- This specifies the maximum number of operations that will be allowed to be held in the work queue before the server starts to reject new requests. A value of zero indicates that there is no limit to the size of the queue.

- `ds-cfg-num-worker-threads` -- This specifies the number of worker threads that will be used to service operations placed in the work queue.

Common Development and Distribution License, Version 1.0

Unless otherwise noted, all components of the OpenDS Directory Server, including all source, configuration, and documentation, are released under the Common Development and Distribution License (CDDL), Version 1.0. The full text of that license is as follows:

```
1. Definitions.

    1.1. "Contributor" means each individual or entity that creates
         or contributes to the creation of Modifications.

    1.2. "Contributor Version" means the combination of the Original
         Software, prior Modifications used by a Contributor (if any),
         and the Modifications made by that particular Contributor.

    1.3. "Covered Software" means (a) the Original Software, or (b)
         Modifications, or (c) the combination of files containing
         Original Software with files containing Modifications, in
         each case including portions thereof.

    1.4. "Executable" means the Covered Software in any form other
         than Source Code.

    1.5. "Initial Developer" means the individual or entity that first
         makes Original Software available under this License.

    1.6. "Larger Work" means a work which combines Covered Software or
         portions thereof with code not governed by the terms of this
         License.

    1.7. "License" means this document.

    1.8. "Licensable" means having the right to grant, to the maximum
         extent possible, whether at the time of the initial grant or
         subsequently acquired, any and all of the rights conveyed
         herein.

    1.9. "Modifications" means the Source Code and Executable form of
         any of the following:

        A. Any file that results from an addition to, deletion from or
           modification of the contents of a file containing Original
           Software or previous Modifications;

        B. Any new file that contains any part of the Original
           Software or previous Modifications; or

        C. Any new file that is contributed or otherwise made
           available under the terms of this License.

    1.10. "Original Software" means the Source Code and Executable
          form of computer software code that is originally released
          under this License.

    1.11. "Patent Claims" means any patent claim(s), now owned or
          hereafter acquired, including without limitation, method,
          process, and apparatus claims, in any patent Licensable by
          grantor.
```