Password Policy Architecture

Version 1.0
July 2006

Contents

Overview

Authentication is often one of the primary functions that an LDAP directory server will be required to perform. Although a number of authentication mechanisms are available, including those using SSL client certificates and Kerberos tickets, most binds use passwords. This includes simple authentication as well as the `DIGEST-MD5`, `CRAM-MD5` and `PLAIN` SASL mechanisms. Passwords offer significantly weaker security than other forms of cryptographic keys, but that can be somewhat mitigated by enforcing strict rules about the use and maintenance of those passwords. This is the role of the password policy.

The password policy features of the OpenDS Directory Server encapsulate those offered by the Sun Java System Directory Server. Where appropriate, they also conform to the specification described in the draft-behera-ldap-password-policy Internet Draft. In particular, revision 09 of this draft has been used for the basis of many client-facing capabilities, and where there may be deviations they will be communicated back for consideration in future revisions.

Copyright and trademark information

Feedback

Please direct any comments or suggestions about this document to:
issues@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*
07/19/06	Initial version.

Password Policy Features

The features provided by the OpenDS password policy include:

- Multiple password policies may be defined in the Directory Server. A default password policy must be defined in the server, which will be used for all entries that do not override this configuration. Additional policies may be defined in the configuration, and user entries may be configured to use those policies either on an individual basis or broadly through virtual attribute capabilities.

- The attribute used to hold the password is customizable. Previous versions of the Directory Server have been hard-coded to store the password in the userPassword attribute. This flexibility makes it possible to use the authentication password attribute syntax as defined in RFC 3112. In the future, it may provide the ability for a user to have multiple passwords in different attributes.

- Password changes may be performed either via standard LDAP modify operations, or through the password modify extended operation as defined in RFC 3062. Some advanced functionality (e.g., the ability for users to change their passwords after they have expired) may require the use of the extended operation. The extended operation also makes it possible to require that users provide their existing passwords before allowing them to specify a new password.

- Passwords may be encoded using one or more password storage schemes. A number of storage schemes are available by default, and an extensible API is provided to allow the creation of additional schemes. Multiple default storage schemes may be defined (so a password can be automatically encoded in several forms for compatibility with applications requiring specific formats), and passwords may also be deprecated so that authenticating with a password stored with a deprecated scheme will cause it to be automatically re-encoded using the default scheme(s).

- Password aging may be used to control the frequency with which passwords are changed. A maximum password age may be defined to enforce password expiration so that passwords must be changed on a regular basis. In addition, a minimum password age may be defined to prevent users from changing passwords too frequently (in particular, in an attempt to circumvent validation features like a password history).

- Password expiration warnings may be provided to clients in the form of response controls for a configurable period of time before the password actually expires to indicate that the password needs to be changed. It is possible to configure the server so that clients must have been provided with at least one warning message before the password will actually be considered expired.

- Once a user's password has expired, that user may be granted one or more grace logins. Limited functionality will be available to the client in this mode of operation so that the only operation that will be allowed by the client will be to change the password.

- Failed authentication attempts may be tracked as a means of preventing brute-force attacks. After a configurable number of failed attempts, the user account may be locked either for a specified duration or until it is manually unlocked by an administrator by resetting the password.

- Users may be automatically required to change their passwords the first time they authenticate after their account is added and/or their password is reset by an administrator. In this case, the only operations that will be allowed for the user will be bind and either a modify operation on the password attribute or the use of the password modify extended operation. Further, a maximum time limit may be imposed for this time period, at which time the user account will be locked and must be reset by an administrator.

- All users associated with a given password policy may be required to change their passwords either immediately or before a given time. This may be used independently of password expiration, or the two capabilities may be used together (e.g., if the expiration interval is to be decreased, this feature may be used to prevent a large portion of user passwords from expiring immediately).

- An extensible password validation API is available for defining sets of criteria that passwords must meet to be allowed for use. For example, password validators may be used to ensure that passwords meet minimum length requirements, that they have characters from different types of character sets, to prevent the re-use of recent passwords, and to prevent the use of common well-known passwords in a specified dictionary. Multiple password validators may be defined for a password policy, in which case a proposed password must be accepted by all of them. It is possible to control whether administrative password changes will be subject to this validation, and it is also possible to reject passwords that are pre-encoded, as that will prevent the validators from checking the clear-text values.

- An extensible password generator API is available that may be used to automatically generate user passwords when they are changed or administratively reset through the password modify extended operation. If a password modify operation is requested but does not provide a new password, then the password generator will be used to supply one. This functionality may be disabled if it is not wanted.

- An extensible account status notification handler API is available that may be used to provide notification to users and/or administrators of significant changes to the state of a user's account. For example, users may be automatically sent an e-mail message the first time that an expiration warning is generated, whenever users change their own passwords, or whenever passwords are reset by an administrator. Similarly, this mechanism could be used to ensure that some action is taken whenever a user's account is locked due to too many failed attempts (e.g., propagate that lockout to other systems).

- The password policy may define basic security requirements for password operations. In particular, it is possible to ensure that bind operations and password changes are performed in a secure manner (either through the use of a secure authentication mechanism or a secure communication channel).

- The password policy may be configured to automatically keep track of the last time that a user authenticated to the Directory Server via a bind operation (access in other forms, like through the use of the proxied authorization control, will not be tracked in this manner). The last login time may be stored in a specified attribute in the user's entry, and it may also be written using a customizable format. The use of the custom format allows for compatibility with existing applications that may require the timestamp to appear in a certain way, and it also may be used to prevent unnecessarily frequent updates to this information (e.g., it may contain only the date of the authentication and not the time, so that a user's account is updated only once per day even if they authenticate multiple times in a day).

- If the password policy is configured to keep track of the last login time, then this information may be used to automatically out an account if it has remained idle for too long. If a user does not authenticate for a specified period of time, then that user's account may be rendered unusable until an administrator manually unlocks that account.

- A user's account may be manually inactivated by an administrator so that it is unable to be used for authentication or authorization, although the user's entry may remain otherwise visible. This operates independently of other password policy features like password expiration or account lockout, and may be used to administratively disable an otherwise active account.

- Many password policy constraints may be taken into account whenever an attempt is made to perform an operation as a user through proxied authorization, or by supplying an alternate authorization ID in the bind operation. In particular, attempts to perform authorization as a user will fail if that user's account is locked, has been inactivated, or the user's password has expired.

The Password Policy Controls

The Directory Server password policy processing makes use of a number of controls to provide various functionality. Those controls include:

- The password policy request control, as defined in the password policy Internet Draft. This request control has an OID of "1.3.6.1.4.1.42.2.27.8.5.1", and it does not have a value. It may be included in any kind of request from the client (with the exception of abandon or unbind requests, as they do not have responses), and is used to request that the server provide a response control with information about the state of the authenticated user's account.

- The password policy response control, as defined in the password policy Internet Draft. This control has an OID of "1.3.6.1.4.1.42.2.27.8.5.1" (the same as the OID for the request control). The value of this control may provide information about a number of properties pertaining to the state of the authenticated user, including both warning and error messages. This control will only be included in response to the corresponding request control.

- The Netscape "password expired" control. This control has an OID of "2.16.840.1.113730.3.4.4", and does not have a value. It is not based on any standard or proposed standard, but has been in use for many years and is supported by a number of clients. It will be included in the bind response for any authentication attempt that fails because the user's password is expired. It may also be included in a successful bind response to indicate that the user's password has been reset by an administrator and must be changed. It will not be included in any response that also includes the password policy response control.

- The Netscape "password expiring" control. This control has an OID of "2.16.840.1.113730.3.4.5", and the value is a string representation of the length of time in seconds until the password actually expires. It is not based on any standard or proposed standard, but has been in use for many years and is supported by a number of clients. It will be included in any successful bind response message during the expiration warning period, unless that bind response includes the password policy response control.

- A Sun-defined account availability request control. This control has an OID of "1.3.6.1.4.1.42.2.27.9.5.8" and does not have a value. It may be included in a search request message to indicate that any search result entry provided in response to that search should include information about the state of the user account associated with that entry.

- A Sun-defined account availability response control. This control has an OID of "1.3.6.1.4.1.42.2.27.9.5.8" (the same as the corresponding request control) and its value indicates whether the account associated with the search result entry is usable (i.e., whether that user will be allowed to authenticate, given the appropriate credentials), as well as additional information regarding the state of that account.

Bind Response Messages

Bind response messages may fall into a number of basic categories:

- The bind response includes a result code indicating that the authentication was successful (i.e., a result code of "0") and does not include any controls. This indicates that the authentication was completely successful with no constraints or additional information.

- The bind response includes a result code indicating that the authentication was successful, but also includes the Netscape "password expiring" control. This indicates that the authentication was completely successful with no constraints, but that the password will expire in the near future.

- The bind response includes a result code indicating that the authentication was successful, but includes the Netscape "password expired" control. This indicates that the authentication was successful, but that the user must change his/her password before performing any other operations.

- The bind response includes a result code indicating that the authentication was successful, and includes the password policy response control. Whether there are any additional constraints depends on the information in the value of that control.

- The bind response includes a result code indicating that a SASL bind is in progress (i.e., a result code of "14"). This will be used for multi-stage SASL bind operations (e.g., `GSSAPI`, `DIGEST-MD5`, or `CRAM-MD5`) and does not directly indicate success or failure.

- The bind response includes a result code indicating that the provided credentials were invalid (i.e., a result code of "49") and does not include any controls. This may indicate that the provided bind DN or authentication info did not refer to a valid user, that the provided credentials were not correct for that user, or that the account is unusable for some reason (e.g., it is locked because of too many failed attempts or has been manually inactivated).

- The bind response includes a result code indicating that the provided credentials were invalid and includes the Netscape "password expired" control. This indicates that the authentication failed because the user's password is expired.

- The bind response includes a result code indicating that the provided credentials were invalid and includes the password policy response control. That control may provide additional information about the reason for the authentication failure.

Bind responses in the OpenDS Directory Server may be significantly different than in previous releases of the Sun Java System Directory Server. The set of result codes will be more limited than

in previous versions, and there will generally not be any textual error messages included in the responses. This will provide better security, because it will avoid disclosing information that may be useful to an attacker (e.g., whether an authentication failure was because the target user didn't exist, because the wrong credentials were provided, or because the account was locked), and it will also help be more standards-compliant.

The Authentication Password Attribute Syntax

The authentication password attribute syntax is defined in RFC 3112 and provides a standardized mechanism for storing encoded password information. Previous versions of the Sun Java System Directory Server have not provided support for storing passwords in this manner, which may prevent or complicate migration of user data from other directory servers that do make use of this capability. However, the authentication password syntax is fully supported in the OpenDS Directory Server.

Passwords encoded in the authentication password syntax consist of five components:

- The name of the password storage scheme. This defines the particular encoding that will be used for the password. RFC 3112 defines two schemes, "`MD5`" and "`SHA1`". The OpenDS Directory Server implements these schemes, as well as others including "`SHA256`", "`SHA384`", and "`SHA512`", which are part of the SHA-2 specification (FIPS 180-2). Additional schemes may be defined using the password storage scheme API.

- A dollar sign character ("`$`"), which is used as a delimiter between the scheme name and the authInfo component.

- An "authInfo" component. The particular meaning of this component may vary based on the password storage scheme, but in all implementations provided with the OpenDS Directory Server, it is a base64-encoded representation of a salt value. The salt value is a set of random bytes that will be appended to the clear-text password in order to make it less vulnerable to dictionary attacks.

- A dollar sign character ("`$`"), which is used as a delimiter between the authInfo and authValue components.

- An "authValue" component. This holds the password, encoded using the specified storage scheme and using the authInfo element as dictated by that scheme. In all OpenDS implementations, the authValue contains a base64-encoded representation of the message digest formed from the concatenation of the clear-text password and the salt bytes, using the algorithm specified in the storage scheme.

A sample authentication password value may look like:

```
SHA1$XkTZSUqgyJk=$gDEnX0QHiCR1M9E2SlJluxE55VU=
```

In this example, the scheme name is "`SHA1`", the authInfo component is "`XkTZSUqgyJk=`", and the authValue component is "`gDEnX0QHiCR1M9E2SlJluxE55VU=`".

The set of authentication passwords available for use in the Directory Server may be retrieved from the `supportedAuthPasswordSchemes` attribute of the root DSE entry. They may also be determined by issuing the command:

```
$ bin/encode-password.sh --listSchemes --authPasswordSyntax
```

The User Password Attribute Syntax

The user password attribute syntax is not strictly based on any standard, but it reflects the format that has historically been used by the Sun Java System Directory Server, as well as some other directories, for storing encoded passwords, and it is referenced in RFC 2307 and the subsequent draft-howard-rfc2307bis Internet Draft.

Passwords encoded in the user password attribute syntax consist of four components:

- An opening curly brace character ("{"), to signify the beginning of the storage scheme name.
- The name of the storage scheme, which defines the manner in which the value is encoded. At the present time, a number of storage schemes are define and available for use, including "CLEAR", "BASE64", "MD5", "SMD5", "SHA", "SSHA", "SSHA256", "SSHA384", and "SSHA512". In the future, additional storage schemes will be added based on reversible encryption, and will likely include at least "3DES", "AES", and "RC4". Support for the UNIX crypt algorithm (a scheme name of "CRYPT") will also be added.
- A closing curly brace character ("}"), to signify the end of the storage scheme name.
- The encoded password value. The way in which the password is encoded is dependent upon the associated storage scheme. For the "CLEAR" scheme, the password is not encoded in any way. For the "BASE64" scheme, the password is simply base64-encoded (which is a very weak form of obfuscation). For the "MD5" and "SHA" schemes, the encoded value is the base64-encoded representation of the message digest for the unsalted clear-text password using the specified algorithm. For the "SMD5", "SSHA", "SSHA256", "SSHA384", and "SSHA512" schemes, the encoded value is the base64-encoded representation of the message digest for the salted clear-text password using the specified algorithm.

A user authentication password value may look like:

```
{SSHA}EZ0rVY46MX/hnBxk4UdRWMARjvqrVTIEJ7zzLg==
```

In this example, the scheme name is "SSHA", and the encoded value is "EZ0rVY46MX/hnBxk4UdRWMARjvqrVTIEJ7zzLg==".

The set of user passwords available for use in the Directory Server may be determined by issuing the command:

```
$ bin/encode-password.sh --listSchemes
```

Password Storage Schemes

Password storage schemes define the ways in which passwords will be encoded for storage in the Directory Server. Password storage schemes must always provide support for the user password syntax, and they may optionally support the authentication password syntax.

The set of password storage schemes defined in the Directory Server exist below the "`cn=Password Storage Schemes,cn=config`" entry in the configuration. All such configuration entries must have the `ds-cfg-password-storage-scheme` objectclass and at least two attributes. The `ds-cfg-password-storage-scheme-class` attribute specifies the fully-qualified name of the Java class that provides the storage scheme implementation, and the `ds-cfg-password-storage-scheme-enabled` attribute indicates whether that storage scheme may be used by entries in the server. Additional configuration attributes may be supported, depending on the storage scheme class, in which case a custom objectclass must also be provided.

Custom password storage schemes may be created by extending the `org.opends.server.api.PasswordStorageScheme` class. This class includes the following methods:

- `initializePasswordStorageScheme` -- Performs any initialization that may be necessary for the password storage scheme, including reading any configuration attributes that may be necessary for use with that scheme.

- `finalizePasswordStorageScheme` -- Performs any cleanup or finalization that may be necessary when the password storage scheme is unloaded.

- `getStorageSchemeName` -- Retrieves the name that will be used for the password storage scheme. This will be the value that appears between the curly braces when it is used in conjunction with the user password syntax.

- `encodePassword` -- Encodes a clear-text password according to the storage scheme. This will contain only the encoded value as it would be used with the user password syntax.

- `encodePasswordWithScheme` -- Encodes a clear-text password according to the storage scheme. The encoded value will include the name of the storage scheme in curly braces, as it would appear with the user password syntax.

- `passwordMatches` -- Indicates whether a provided clear-text password matches a value that has been encoded with the storage scheme.

- `supportsAuthPasswordSyntax` -- Indicates whether the password storage scheme supports the use of the authentication password syntax.

- `getAuthPasswordSchemeName` -- Retrieves the name of the password storage scheme as it should be used in conjunction with the authentication password syntax.

- `encodeAuthPassword` -- Encodes a clear-text password according to the authentication password syntax. The resulting password will be encoded with the scheme name, authInfo, and authValue components.

- `authPasswordMatches` -- Indicates whether a provided clear-text password matches a value that has been encoded with the storage scheme according to the authentication password syntax.

- `isReversible` -- Indicates whether the password storage scheme encodes values in a reversible manner that makes it possible to obtain the original plaintext value from the encoded value.

- `getPlaintextValue` -- Retrieves the plaintext value for the provided stored password.

- `isStorageSchemeSecure` -- Indicates whether the password storage scheme encodes passwords in a manner that may be considered secure. A password may be considered secure if it is not possible to determine the plaintext value from the encoded value, or if it is only possible to determine the plaintext value with a secret key.

Password Validators

Password validators provide a mechanism to determine whether a provided plaintext password is acceptable for use. They can be used to help prevent users from choosing trivial passwords that are weak and may be easily-guessed. Potential types of validation that may be performed include:

- Ensuring that a password has at least a specified minimum number of characters.
- Ensuring that a password has no more than a specified maximum number of characters.
- Ensuring that a password contains at least a specified number of characters from different character sets (e.g., lowercase letters, uppercase letters, numeric digits, and symbols).
- Ensuring that a user is not allowed to re-use a password that has been previously used by that user (i.e., that the password is not contained in a history of previous passwords).
- Ensuring that a user is not allowed to choose a password that matches the value of another attribute in the user's entry.
- Ensuring a password is not contained in a specified dictionary.

The set of password storage schemes defined in the Directory Server exist below the "`cn=Password Validators,cn=config`" configuration entry. Those entries must contain the `ds-cfg-password-validator` objectclass, the attribute `ds-cfg-password-validator-class` attribute to indicate the name of the class providing the password validator implementation, and the `ds-cfg-password-validator-enabled` attribute to indicate whether the validator may be used in the server. Additional configuration attributes may be used to provide more information for use by individual validator implementations.

Custom password validators may be created by extending the `org.opends.server.api.PasswordValidator` class, which includes the following methods:

- `initializePasswordValidator` -- Performs any necessary initialization for the password validator.
- `finalizePasswordValidator` -- Performs any finalization that may be necessary for the password validator.
- `passwordIsValid` -- Indicates whether the provided clear-text password is acceptable according to the password validator.

Password Generators

As the name implies, password generators are used to generate passwords for user accounts. They will be used in the course of processing a password modify extended operation. If no new password is provided, then a password generator will be used to create one.

Note that because a password policy may have a configurable set of password validators, including custom validator, the set of criteria used to determine whether a password is acceptable may vary. As such, passwords created by a password generator will not be subject to validation. However, it is recommended that password generators be configured so that the passwords they create will be in-line with the requirements of the validators.

Password generators are defined in the Directory Server configuration below the "`cn=Password Generators,cn=config`" entry. These entries must have the `ds-cfg-password-generator` objectclass and the `ds-cfg-password-generator-class` and `ds-cfg-password-generator-enabled` attributes. Additional configuration attributes may be available, depending on the password generator implementation.

Custom password generators may be created by extending the `org.opends.server.api.PasswordGenerator` API. This includes the following methods:

- `initializePasswordGenerator` -- Performs any necessary initialization for the password generator whenever it is loaded.
- `finalizePasswordGenerator` -- Performs any finalization that may be necessary for the password generator when it is unloaded.
- `generatePassword` -- Generates a password for use in a user entry.

Account Status Notification Handlers

Account status notification handlers are used to provide notification of significant events that may happen in the course of password policy processing. The events that may generate an account status notification include:

- An account has been temporarily locked as a result of too many failed authentications.
- An account has been permanently locked as a result of too many failed authentications.
- An authentication attempt fails because the account has remained idle for too long.
- An authentication attempt fails because the user's password has expired.
- The first time that a user is warned about an upcoming password expiration.
- A user's password has been reset by an administrator.
- A user's password is changed by that user.
- An account has been manually inactivated by an administrator.
- An account has been manually re-activated by an administrator.

Account status notification handlers are defined below the "`cn=Account Status Notification Handlers,cn=config`" configuration entry. Those entries should contain the `ds-cfg-account-status-notification-handler` objectclass and the `ds-cfg-notification-handler-class` and `ds-cfg-notification-handler-enabled` configuration attributes. Additional configuration attributes may be used depending on the notification handler implementation.

By default, the OpenDS Directory Server will provide an account notification handler capable of sending e-mail messages to end users and/or administrators, as well as one that will write the notifications to the server's error log. Additional notification handlers may be created by extending the `org.opends.server.api.AccountStatusNotificationHandler` class, which includes the following methods:

- `initializeStatusNotificationHandler` -- Performs any initialization that may be required when the account status notification handler is loaded.

- `finalizeStatusNotificationHandler` -- Performs any cleanup that may be needed whenever the notification handler is unloaded.

- `handleStatusNotification` -- Performs any work that may be necessary when an account status notification is generated.

SASL Mechanism Handlers

SASL mechanism handlers are components that allow the Directory Server to process certain types of SASL bind requests (where SASL refers to the Simple Authentication Security Layer and is defined in RFC 2222). The Directory Server currently provides support for a number of SASL mechanisms, including:

- `ANONYMOUS` -- This is used to perform anonymous authentication to the server. The only notable difference between the `ANONYMOUS` SASL mechanism and using simple authentication to bind anonymously (by providing an empty password) is that the `ANONYMOUS` SASL mechanism also provides the ability to include trace information in the request that can be used to associate it with a particular type of client. However, because there is no real authentication, this trace information should only be considered useful for debugging purposes and should not actually be trusted.

- `CRAM-MD5` -- This is used to perform password-based authentication to the Directory Server in a manner that does not expose the password to an observer and therefore does not compromise authentication security even over an unencrypted communication channel, although it does require that the Directory Server have access to the plaintext password (stored in the clear or using a reversible encryption algorithm). It provides weaker security than the `DIGEST-MD5` mechanism, but is supported by a large number of clients.

- `DIGEST-MD5` -- This is very similar to the `CRAM-MD5` mechanism in that it allows for password-based authentication without exposing the password itself. It is more secure than the `CRAM-MD5` mechanism because the authentication performed using `DIGEST-MD5` includes random data from both the client and the server whereas the authentication performed using `CRAM-MD5` includes random data only from the server (which is easier to exploit via a man-in-the-middle attack). As with the `CRAM-MD5` mechanism, the use of `DIGEST-MD5` requires that the Directory Server have access to the plaintext user password, so it must be stored in the clear or using a reversible encryption algorithm.

- `EXTERNAL` -- This is used to authenticate to the Directory Server using information that is available outside of the context of the protocol communication. In the OpenDS Directory Server, the only source of external authentication information that may be used is an SSL client certificate. That is, if a client provides its own SSL certificate to the Directory Server in the course of establishing a secure communication channel, then the information in that certificate may be used to identify the client.

- `GSSAPI` -- This is used to authenticate to the Directory Server using a Kerberos V ticket that the client has obtained from the KDC. This can be used to provide a kind of single sign-on for Directory Server clients.

- `PLAIN` -- This is used to perform password-based authentication to the Directory Server. There is no inherent security provided by this mechanism (i.e., it does not obscure the

password in any way like with the CRAM-MD5 or DIGEST-MD5 mechanisms), and in this regard it is very similar to performing simple authentication. The only real difference between the use of simple authentication and the PLAIN SASL mechanism is that the PLAIN SASL mechanism allows the user identity to be provided as an authentication ID (e.g., a user name) rather than requiring the full user DN as is necessary with simple authentication.

The set of SASL mechanisms available for use are defined below the "cn=SASL Mechanisms,cn=config" entry in the Directory Server configuration. All such entries must have the ds-cfg-sasl-mechanism-handler objectclass, as well as the ds-cfg-sasl-mechanism-handler-class and ds-cfg-sasl-mechanism-handler-enabled attributes. Additional attributes may be available on a per-mechanism basis.

Additional SASL mechanisms may be defined in the server by extending the org.opends.server.api.SASLMechanismHandler class, which includes the following methods:

- initializeSASLMechanismHandler -- Performs any necessary initialization for the SASL mechanism handler.

- finalizeSASLMechanismHandler -- Performs any necessary finalization for the SASL mechanism handler.

- processSASLBind -- Performs the actual work of processing the SASL bind request. Note that for certain SASL mechanisms, it may be necessary to have multiple request/response cycles in order to perform all necessary processing, in which case it will be necessary to maintain state information in the client connection to make it possible to resume the authentication process.

- isPasswordBased -- Indicates whether this SASL mechanism is based on the use of passwords for authentication. This will be used by the password policy to determine whether certain constraints should be enforced (e.g., a mechanism that does not use a password to authenticate should not be subject to password expiration).

- isSecure -- Indicates whether this SASL mechanism is secure, in that it does not expose the user credentials in a manner that may be exploited by an observer.

Password Policy Configuration

As with previous versions of the Sun Java System Directory Server, multiple password policies may be defined. One policy will be designated the default policy, but additional policies may be defined and applied to users on an individual basis. Unlike previous versions, however, all password policy definitions must exist in the Directory Server configuration under the "`cn=Password Policies,cn=config`" entry -- they may not be defined as subentries in the directory data. This allows for better separation between configuration and data, and also makes it easier to have different password policy configurations in different sets of servers.

Another notable change that we will enforce for the OpenDS Directory Server that varies from previous releases is that each password policy configuration entry will be completely independent. In current versions of the Sun Java System Directory Server, there is a kind of hierarchy such that if a password policy does not specify a value for a given configuration attribute it will be inherited from the default password policy, or if it is the default policy then it will use a predefined default value. This can be a confusing behavior because it is not documented anywhere (except within the code itself), and it can produce results that are different from what the administrator might expect. Further, it would increase the complexity of the OpenDS implementation, and it would cause problems with certain multivalued configuration attributes like those that specify the password validators or account status notification handlers. It would also cause notable problems if the default policy was changed while the server was online. As a result, all OpenDS password policy configuration entries will be considered independent and there will not be any form of inheritance between them.

The `ds-cfg-default-password-policy` configuration attribute in the `cn=config` entry holds the DN of the configuration entry for the default password policy to be used in the server. This entry must exist, and it will define the password policy that will be used for entries that do not explicitly specify which policy to use.

As noted above, all password policy configuration entries must reside below the "`cn=Password Policies,cn=config`" entry. Password policy configuration entries must contain the `ds-cfg-password-policy` objectclass. This differs from the `pwdPolicy` objectclass defined in the Internet Draft in the following ways:

- It includes configuration attributes for a number of features that are not covered by the draft.
- It excludes configuration attributes like `pwdInHistory` and `pwdMinLength` that pertain to password validation and therefore are more appropriate in password validator configuration entries.
- The names of many of the configuration attributes have been changed to make their function clearer.

The `ds-cfg-password-policy` objectclass defines the following configuration attributes:

- `ds-cfg-password-attribute` -- This specifies which attribute should be used to hold user passwords. The specified attribute type must be defined in the server schema, and it must have either the user password or the authentication password attribute syntax. This is a required attribute that must be present in all password policy definitions.

- `ds-cfg-default-password-storage-scheme` -- This specifies the names of the default storage scheme(s) that will be used to encode passwords for the password policy. The names of the schemes should be in accordance with the password syntax (i.e., if the password attribute has the user password syntax, then the user password scheme names should be used). This is a multi-valued attribute, if multiple storage schemes are defined then passwords will be automatically encoded using all of those schemes. This is a required attribute, and at least one default password storage scheme must be configured in all password policy definitions.

- `ds-cfg-deprecated-password-storage-scheme` -- This specifies the names of the password storage scheme(s) that should be considered deprecated. If a user authenticates to the Directory Server and that user's password is encoded using one of these schemes then that password will be re-encoded using the default schemes.

- `ds-cfg-password-validator-dn` -- This specifies the DNs of the configuration entries for the password validators that should be used with the password policy. This is a multi-valued attribute, and if multiple password validators are specified then all of them must accept a proposed password for it to be allowed.

- `ds-cfg-skip-validation-for-administrators` -- This indicates whether password validation will be performed for passwords supplied by administrators. If this is set to "`true`", then passwords supplied by administrators when adding an entry or resetting a user's password will not be subject to validation checks. If it is set to "`false`", or if this attribute is not defined for the password policy, then passwords provided by administrators will be subject to the same validation that will be required for user password changes.

- `ds-cfg-account-status-notification-handler-dn` -- This specifies the DNs of the configuration entries for the account status notification handlers that should be used with the password policy. This is a multi-valued attribute, and if multiple notification handlers are configured, then all of them will be used whenever an appropriate event occurs.

- `ds-cfg-password-generator-dn` -- This specifies the DN of the configuration entry for the password generator for use with the password policy. This is a single-valued attribute, and if no password generator is specified then the server will not provide the ability to generate a password for a user if none is defined.

- `ds-cfg-allow-user-password-changes` -- This indicates whether end users will be allowed to change their own passwords. Note that this works in conjunction with access control definitions, and for users to be allowed to change their own passwords, it must be

allowed both by this configuration attribute and by the access controls defined in the server.

- `ds-cfg-password-change-requires-current-password` -- This indicates whether users will be required to provide their current passwords in order to choose a new password. This can add an additional layer of security, and it may also be useful in conjunction with password validators (e.g., to ensure that the new password is sufficiently different from the previous password). If users are required to provide their current password, then they will be required to use the password modify extended operation to change their passwords.

- `ds-cfg-allow-multiple-password-values` -- This indicates whether users will be allowed to have multiple different passwords in the same password attribute. Although this is technically allowed by the specification, it is not recommended because it is difficult to maintain and will weaken the security of the Directory Server. Note that this does not apply to a single password that is encoded using multiple storage schemes.

- `ds-cfg-allow-pre-encoded-passwords` -- This indicates whether password changes will be allowed to provide the passwords in a pre-encoded form. In most cases, this is undesirable because the Directory Server will not be able to determine the plaintext password in order to verify whether it meets the password validation criteria.

- `ds-cfg-maximum-password-age` -- This specifies the maximum length of time that a user will be allowed to keep the same password (i.e., the password expiration interval). The value will be specified as an integer followed by a time unit of "`seconds`", "`minutes`", "`hours`", "`days`", or "`weeks`". A value of "`0 seconds`" indicates that no password expiration will be enforced.

- `ds-cfg-password-expiration-warning-interval` -- This specifies the length of maximum length of time before the password expires that the server should start providing warning indicators to the client in the form of the Netscape password expiring control or the password policy response control. This warning will be used if the upcoming expiration is due to the password maximum age, the maximum reset age, or a forced password change. The value will be specified as an integer followed by a time unit of "`seconds`", "`minutes`", "`hours`", "`days`", or "`weeks`". A value of "`0 seconds`" indicates that no warning will be provided.

- `ds-cfg-expire-passwords-without-warning` -- This indicates whether a user's password will be allowed to expire if that user has not yet received any warning notification (e.g., because the user has not authenticated during the warning interval). If this option is set to "`true`", then the user's password will be expired regardless of whether that user has seen any warning indications. If this option is set to "false", then the time that the password will actually expire will be equal to the time of the first warning notification plus the duration of the warning interval.

- `ds-cfg-allow-expired-password-changes` -- This indicates whether users will be allowed to change their own passwords even after their password has expired. If this is set to "`true`", then users will be allowed to reset expired passwords using the password modify extended operation.

- `ds-cfg-grace-login-count` -- This specifies the maximum number of grace logins that a user will be allowed. Grace logins allow users to authenticate to the Directory Server after their passwords have expired for the purpose of changing their passwords. The only operation that a user will be allowed in this mode is to change the password.

- `ds-cfg-minimum-password-age` -- This specifies the minimum length of time that a user will be required to keep the same password. This can prevent users from changing their passwords too quickly in an attempt to subvert a password policy feature (e.g., quickly changing the password several times to flush an old password from the history so it may be re-used). The value will be specified as an integer followed by a time unit of "`seconds`", "`minutes`", "`hours`", "`days`", or "`weeks`". A value of "`0 seconds`" indicates that there will not be a minimum age.

- `ds-cfg-lockout-failure-count` -- This specifies the maximum number of consecutive failed authentication attempts that a user will be allowed before the account is locked. After this number of failed attempts, the user will be unable to authenticate even if the correct password is provided. Note that lockout due to failed attempts may not be available for SASL mechanisms because the identity of the user may not be available for an unsuccessful authentication. A value of zero indicates that account lockout will not be enforced.

- `ds-cfg-lockout-failure-count-expiration-interval` -- This specifies the length of time that a failed bind attempt will be counted toward the total lockout failure count. The record of failed authentication attempts will automatically be cleared if the user is able to bind successfully, but failed authentications may also be forgiven after a period of time if the account is not locked (it will not cause the account to be unlocked). The value will be specified as an integer followed by a time unit of "`seconds`", "`minutes`", "`hours`", "`days`", or "`weeks`". A value of "`0 seconds`" indicates that failed authentications will not be allowed to expire.

- `ds-cfg-lockout-duration` -- This specifies the length of time that a user's account will be locked as a result of failed authentications. After this length of time has elapsed, the user will again be allowed to authenticate and the previous authentication failures will not be counted against the user. The value will be specified as an integer followed by a time unit of "`seconds`", "`minutes`", "`hours`", "`days`", or "`weeks`". A value of "`0 seconds`" indicates that the lockout will be permanent, until the password is reset by an administrator.

- `ds-cfg-force-change-on-add` -- This indicates whether users will be required to change their passwords the first time that they authenticate after their account has been added. Note that this applies only to add operations performed over protocol and not to entries

created by LDIF import. In this mode, a user will be allowed to authenticate but will be required to change the password before any other operation may be performed.

- `ds-cfg-force-change-on-reset` -- This indicates whether users will be required to change their passwords after they have been reset by an administrator (where "administrator" is defined as anyone that changes a user's password other than the user whose password has been reset). In this mode, a user will be allowed to authenticate but will be required to change the password before any other operation may be performed.

- `ds-cfg-maximum-password-reset-age` -- This specifies the maximum length of time that users will be given to change their passwords after they have been reset by an administrator. If users are required to change their passwords after they have been reset by an administrator, then it may be desirable to ensure that users authenticate and choose a new password shortly after it is reset to minimize the chance that it will be accessed by an unauthorized user (e.g., if the temporary password provided by the administrator may be weak or communicated to the user using an insecure means). Once this time period has elapsed, the account will be locked and the password will again need to be reset by an administrator before it can be used. The value will be specified as an integer followed by a time unit of "`seconds`", "`minutes`", "`hours`", "`days`", or "`weeks`". A value of "`0 seconds`" indicates that no time limit will be enforced for this operation.

- `ds-cfg-require-secure-authentication` -- This indicates whether users with the password policy will be required to authenticate to the Directory Server in a secure manner (i.e., using a mechanism that does not provide an observer with enough information to impersonate that user). If this is set to "`true`", then users will be required to authenticate using a mechanism that does not expose the clear-text credentials (e.g., the DIGEST-MD5 or CRAM-MD5 SASL mechanisms), or authenticate over an encrypted channel (e.g., using a connection encrypted by SSL or using the StartTLS extended operation).

- `ds-cfg-require-secure-password-changes` -- This indicates whether users with the password policy will be required to change their passwords in a secure manner (i.e., one that does not expose the new password to an observer). If this is set to "`true`", then users will be required to use a secure communication channel when changing their passwords, or the new password must be pre-encoded using a secure storage scheme (which will only be allowed if the `ds-cfg-allow-pre-encoded-passwords` option is set to "`true`", in which case the new password will not be subject to validation). This will apply to administrative password reset operations as well as user password changes.

- `ds-cfg-require-change-by-time` -- This specifies a time (using the generalized time attribute syntax) by which all users with the password policy will be required to change their passwords. If this is set to a time in the future, then it will be treated like a password expiration constraint, causing the Directory Server to generate a password expiration warning notice for authentications that occur within the warning period before this time arrives. If it is set to the current time or a time in the past, then users will be allowed to authenticate, but they will be required to change their passwords before any other operation can be performed.

- `ds-cfg-last-login-time-attribute` -- This specifies the name of an attribute type that will be used to hold the last login time for users with this password policy. This should be an attribute that is defined in the server schema with a syntax that will allow the time string. It must also be an attribute type that is either allowed by the objectclasses contained in user entries, or it must be an operational attribute (in which case, schema checking will not be enforced). Whenever a user authenticates to the Directory Server using a bind operation, the entry will be updated so that the value of this attribute will be set to a current timestamp. If no attribute type is specified, then the last login time will not be maintained.

- `ds-cfg-last-login-time-format` -- This specifies a format string that defines the manner in which the last login time should be stored in a user's entry. This may be any valid format string that is allowed for use with the `java.text.SimpleDateFormat` object. This format string will make it possible to store the value in a manner that is compatible with other applications that may need to use it. It may also be used to limit the frequency of updates due to authentication, as the user's entry will only actually be modified if the calculated last login time string differs from the value that the attribute currently has (e.g., if this format string only holds information about the date but not the time of the authentication, then a user's entry will only be updated with a new last login time at most once a day, regardless of the number of times that the user authenticates over the course of that day). If no format string is provided, then the last login time will not be maintained.

- `ds-cfg-previous-last-login-time-format` -- This specifies one or more format strings that may have been used for maintaining the last login time in the past. The sole purpose of this attribute is to make it possible for last login times written with an earlier format to be parsed so that they can be used in conjunction with the idle lockout feature.

- `ds-cfg-idle-lockout-duration` -- This specifies the maximum length of time that a user's account may remain idle before it will automatically be locked so that it will not be allowed to authenticate. This feature will only be available if the last login time is maintained for users. Any user that has not authenticated to the Directory Server for a period of at least this length of time will not be allowed to authenticate until their password has been reset by an administrator. The value will be specified as an integer followed by a time unit of "`seconds`", "`minutes`", "`hours`", "`days`", or "`weeks`". A value of "`0 seconds`" indicates that accounts will not be locked because they have remained idle for too long.

Password Policy Operational Attributes

In order to provide certain password policy features, it will be necessary to store some state information in user entries. This information will be stored using operational attributes, and in general they will be declared with the `NO-USER-MODIFICATION` flag to prevent them from being directly altered by end users or administrators.

Wherever possible, the operational attributes defined in the IETF password policy draft will be used for the sake of compatibility with other directory server implementations that may also support the same specification. Unless otherwise noted, any attribute name that begins with "`pwd`" should be assumed to come from this password policy draft, and its meaning should be consistent with its definition in that specification.

The operational attributes that the Directory Server will maintain in a user's entry include:

- `pwdChangedTime` -- This holds the timestamp (in generalized time format) of the last time that the user's password was changed, either by that user or by an administrator. It will automatically be set on an add, modify, or password modify operation that sets or alters the user's password, and it should never be cleared or unset. It will be used to determine when the user's password was last changed for the purposes of enforcing the minimum and maximum password ages, and to determine whether to generate expiration warning notifications. It will also be used in conjunction with the `pwdReset` attribute to enforce the maximum password reset age.

- `pwdGraceUseTime` -- This holds the timestamps (in generalized time format) of the times that a user authenticated with a grace login after that user's password had expired, in order to ensure that the maximum number of grace logins will be enforced. This will automatically be set whenever the user authenticates using one of the grace logins, and it will be cleared whenever the user's password is changed by that user or reset by an administrator.

- `pwdFailureTime` -- This holds the timestamps (in generalized time format) of the times that an authentication attempt failed for the user because the wrong password was provided. It will be used to enforce the maximum failure account, so that an account may be locked as a result of too many failed attempts. This will be set automatically whenever such an authentication failure occurs, and will be cleared whenever the user authenticates successfully (whether before the lockout occurs or after the account has been locked and the lockout duration has passed) or whenever the user's password is changed by that user or reset by an administrator.

- `pwdAccountLockedTime` -- This holds the timestamp (in generalized time form) of the time that the user's account was locked as a result of too many failed authentication attempts. It will be used to indicate that the account is locked, as well as to provide information about when it may be automatically unlocked via the password lockout duration. It will be automatically cleared if the user's password is reset by an administrator, or on any authentication attempt (regardless of its success or failure) after the lockout duration has passed.

  Note that the OpenDS password policy implementation does vary from the behavior specified in the password policy draft in one significant way. In the OpenDS implementation, this attribute will always hold the time that the account was locked, regardless of whether the account lockout will be temporary or permanent. The password policy draft states that in the event that the account should not be automatically unlocked after some period of time, it should be given a special value of "`00000101000000Z`". There are several justifications for this variation, but the primary reasons are that the time specified in the draft is actually illegal (the Gregorian calendar does not have a year "0"), and this special value is unnecessary because the determination about whether the account is locked temporarily or permanently may be made based on the value of the `ds-cfg-lockout-duration` attribute (a value of "`0 seconds`" indicates that the account should not be automatically unlocked).

- `pwdPolicySubentry` -- This holds the DN of the configuration entry for the password policy that should be enforced for the associated user. If this is not defined, then the user will be subject to the default password policy for the Directory Server. If it is defined, then it must refer to a valid password policy definition that exists below the "`cn=Password Policies,cn=config`" configuration entry. If this attribute exists in a user's entry but does not refer to a valid entry (or if it refers to any entry that is not below "`cn=Password Policies,cn=config`"), then the user will not be allowed to authenticate. This attribute will never be automatically set or cleared by the Directory Server, but must be manually specified by the administrator, or may be generated as a virtual attribute.

- `pwdReset` -- This holds a Boolean value of "`true`" if the user's password has been reset by an administrator and must be changed before the user will be allowed to perform any other kind of operation. It will be automatically set to "`true`" when the user's account is added if the `ds-cfg-force-change-on-add` attribute is set to "`true`", or on an administrative modify or password modify operation that resets the user's password if the `ds-cfg-force-change-on-reset` attribute is set to "`true`". It will be automatically cleared whenever the user's password is changed by that user.

- `ds-pwp-account-disabled` -- This holds a Boolean value of "`true`" if the user's account has been manually disabled by an administrator, in which case that user will not be allowed to authenticate to the Directory Server. This attribute will never be automatically set or cleared by the Directory Server, but must be manually specified by the administrator, or may be generated as a virtual attribute.

- `ds-pwp-last-login-time` -- This attribute is provided for use as the default attribute for holding last login time information if that feature should be enabled. If that feature is

enabled, then there is no requirement that this attribute be used, and an alternate attribute may be configured if the administrator so chooses.

- `ds-pwp-password-changed-by-required-time` -- This attribute may hold a generalized time value that is equal to the value of the `ds-cfg-require-change-by-time` attribute in the password policy configuration entry. It is used to indicate whether the user's password has been changed in accordance with that configuration. This attribute will automatically be set to the value of the `ds-cfg-require-change-by-time` attribute whenever the user's password is changed (by the end user or an administrator) any time that configuration attribute has a value that is different from the value currently held in the `ds-pwp-password-changed-by-required-time-time` attribute.

- `ds-pwp-warned-time` -- This attribute holds a timestamp (in generalized time form) that indicates when the user was first warned about an upcoming password expiration. It will be used in conjunction with the `ds-cfg-expire-passwords-without-warning` configuration attribute to determine whether a user has seen an expiration warning and if so what the new adjusted expiration time should be. It will be automatically set by the Directory Server the first time that a warning notification is sent to indicate that a password is about to expire, and it will be cleared whenever the user's password is changed (either by that user or an administrator).

It is important to note that the set of operational attributes used for maintaining password policy state information should be kept relatively small to avoid bloating user entries, and most of these attributes will only be present under certain conditions. However, it is also important to avoid overloading the values of these attributes so that they may be evaluated in different ways depending on the particular value that they are assigned. This has been a problem in previous versions of the Sun Java System Directory Server, in which the `passwordExpirationTime` operational attribute could have a number of different meanings (e.g., it could indicate the time that a user's password was supposed to expire, or to indicate that the user's password should never expire, or to indicate that the user's password had been reset by an administrator). Having multiple meanings for the same attribute will generally lead to problems and should be avoided at all costs. Note, however, that it may be considered acceptable for multiple components to use the same attribute if they are all using it in exactly the same way.

Client Connection State Variables

In order to ensure that password policy processing does not impose a significant performance penalty, certain state information will be maintained in variables associated with the client connection structure. This is a minimal set of variables, including:

- `authenticationInfo` -- This provides information about the current authentication state for the client connection. It is a structure that indicates whether the client is authenticated and if so holds the associated authentication and authorization DNs. It also provides information about the manner in which the client authenticated and whether that client should be considered a root user.

- `bindInProgress` -- This indicates whether there is currently a bind operation in progress on the client connection. If a bind operation is in progress, then no other operation will be allowed until the bind processing has been completed.

- `mustChangePassword` -- This indicates whether the user associated with the client connection will be required to change his/her password before being allowed to perform any other operation.

- `saslAuthState` -- This is used to hold opaque state information used in processing a multi-stage SASL bind. For example, for a SASL bind using the `DIGEST-MD5` mechanism it may hold information about the nonce that was selected by the server.

Implementation Details

In the course of password policy processing, there will be several core operations that will need to be performed that will require interacting with the password policy configuration and/or state information in the user's account. This section will detail the most common kinds of determinations that will need to be made and the underlying logic that will be used to make those decisions.

Determining a User's Password Changed Time

In virtually all cases, the password changed time for a user will always be contained in the `pwdChangedTime` operational attribute. This attribute will automatically be set whenever an entry containing a password is added to the server over protocol, and it will also be set whenever a user's password is changed by that user or is reset by an administrator. As such, if the `pwdChangedTime` attribute is set, then it should be used to determine the time the user's password was last changed.

If a user's account does not include a changed time for some reason (which may occur, for example, if the user's entry was added during an LDIF import an the LDIF representation did not include this attribute), then the Directory Server will attempt to fall back on the `createTimestamp` operational attribute if it is present in the user's entry. If that attribute is not present, then the current time should be used, and the user's entry should be updated to store this value in the `pwdChangedTime` operational attribute.

Determining a User's Last Login Time

If last login time tracking is enabled for the Directory Server, then the attribute specified in the value of the `ds-cfg-last-login-time-attribute` configuration attribute should hold this value, in the format specified in the `ds-cfg-last-login-time-format` attribute. If this attribute is present in the user's entry and it is in a form that can be parsed using that last login time format, then it will be used as the last login time for the user.

If the last login time attribute exists, but its value cannot be parsed using the format string provided in the `ds-cfg-last-login-time-format` configuration attribute, then the Directory Server should determine whether there are any values for the `ds-cfg-previous-last-login-time-format` configuration attribute, and if so whether the value of the last login time attribute may be parsed using any of those formats.

If the last login time attribute does not exist in a user's entry, or if its value cannot be parsed using the current format or any of the previous formats, then the Directory Server should attempt to use the value of the `createTimestamp` operational attribute. If all else fails, it should be assumed that

user has never authenticated to the Directory Server (or at least has never done so with last login time tracking enabled).

Note that there is one notable exception to this practice: for the purposes of enforcing idle lockout, if the difference between the current time and the last login time is greater than the idle lockout duration, then the value of the `pwdChangedTime` operational attribute may be used in place of the last login time (only this attribute will be allowed -- it will not fall back on the `createTimestamp` or default to the current time). This will allow an administrative password reset to unlock a user's account as a result of the idle lockout.

Determining Whether a User's Account Is Disabled

In order to determine whether a user account is disabled (i.e., inactivated so that the specified user should not be allowed to authenticate), it is only necessary to look at the user's entry to retrieve the `ds-pwp-account-disabled` attribute. If this attribute exists in the user's entry with any value other than "`false`", then the account will be disabled. If the attribute does not exist or has a value of "`false`", then the account will not be disabled.

Determining Whether a User's Password Has Been Reset

The process for determining whether a user's password has been reset is primarily based on the `pwdReset` operational attribute (which is automatically set in a user's entry whenever the password is reset by an administrator and the "force change on reset" option is enabled, or when the account is added to the server if the "force change on add" option is enabled). If this attribute is present with a value of anything other than "`false`", then the account has been placed in a reset mode, although if the maximum password reset age is nonzero then the user's account may be locked if it has been longer than this time period since the password was reset.

The other way that a user's account may enter a reset mode can occur if an administrator has specified a required change time in the `ds-cfg-require-change-by-time` attribute of the password policy. If there is a required change time and it is in the past, then a user's account will be placed in reset mode unless the user's entry contains the `ds-pwp-password-changed-by-required-time` operational attribute with a value exactly equal to the required change time in the password policy definition. If the user's account is in this mode, then it will not be subject to the maximum password reset age.

The processing required to make this determination can be described by the following pseudocode:

```
boolean inResetMode = false;
```

```
if ((pwdResetValue != null) && (pwdResetValue != "false"))
{
  if (maxResetAge > 0)
  {
    if ((passwordChangedTime + maxResetAge) < currentTime)
    {
      inResetMode = true;
    }
  }
  else
  {
    inResetMode = true;
  }
}

if ((requireChangeTime > 0) && (requireChangeTime < currentTime) &&
    (userRequiredChangeTime != requireChangeTime))
{
  inResetMode = true;
}

return inResetMode;
```

Determining Whether a User's Account is Locked

There are two possible reasons that a user's account may be locked:

- It may be locked as a result of too many consecutive failed authentication attempts, in which case the `pwdAccountLockedTime` operational attribute will be set in the user's entry to the time that the account was locked. This lockout may be temporary or permanent, depending on whether the `ds-cfg-lockout-duration` configuration attribute has a nonzero value. If there is a nonzero lockout duration, then it will also be necessary to calculate the time the account should be unlocked to determine if the lockout is still in effect.

- It may be locked as a result of the user account remaining inactive for too long. This lockout will always be permanent, and it requires that the last login time feature be enabled in the password policy as well as a nonzero value for the idle lockout duration. Note that an administrative password reset can override the idle account lockout, so it should be checked as well if the account would otherwise be unlocked.

The following pseudocode outlines the basic processing that should be performed to determine if a user's account is locked:

```
boolean isLocked = false;

if (accountLockedTime > 0)
```

```
{
  if (lockoutDuration > 0)
  {
    if ((accountLockedTime + lockoutDuration) > currentTime)
    {
      isLocked = true;
    }
  }
  else
  {
    isLocked = true;
  }
}

if ((idleLockoutDuration > 0) && (lastLoginTime > 0))
{
  if ((currentTime - lastLoginTime) > idleLockoutDuration)
  {
    if ((currentTime - pwdChangedTime) > idleLockoutDuration)
    {
      isLocked = true;
    }
  }
}

return isLocked;
```

Determining a User's Password Expiration Time

The process for determining the time that a user's password should be considered expired is a somewhat complex task because it needs to take several factors into account, including the password maximum age, the maximum reset age, the forced change time, and the expire without warning flag. A basic set of pseudocode for determining the length of time until expiration is as follows:

```
expirationTime = NEVER;

if (maxAge > 0)
{
  expTime = passwordChangedTime + maxAge;
  if ((expTime < currentTime) && (! expirePasswordsWithoutWarning))
  {
    if (userHasBeenWarned)
    {
      expTime = warnedTime + warningInterval;
    }
    else
    {
      expTime = currentTime + warningInterval;
    }
  }
```

```
  expirationTime = Math.min(expirationTime, expTime);
}

if ((maxResetAge > 0) && userPasswordIsReset)
{
  expTime = passwordChangedTime + maxResetAge;
  expirationTime = Math.min(expirationTime, expTime);
}

if ((requireChangeTime > currentTime) &&
    (requireChangeTime != userRequireChangeTime))
{
  expTime = requireChangeTime - currentTime;
  expirationTime = Math.min(expirationTime, expTime);
}

return expirationTime;
```

If the above code returns a time that is less than or equal to the current time, then the user's password should be considered expired. If the above code returns a time that is greater than the current time plus the warning interval, then the user's password should not be considered expired, and there is no need to send a warning notice. Otherwise, the user's password is not expired, but it will expire in the near future and a warning notice should be sent.

Note that each of these three conditions is handled a little differently, and it is important to understand the subtle differences. They are:

- When determining the expiration time relative to the maximum password age, it may be necessary to take the "expire without warning" feature into account. If the password is older than the maximum allowed age but the "expire without warning" configuration option is set to "`false`" (which is the default setting), then the expiration time should be recalibrated to be the time of the first warning plus the length of the warning interval (or the current time plus the length of the warning interval if no warning has yet been sent).

- When determining the expiration time relative to the maximum password reset age, it is only applicable if the user's account is in a reset state. If this is the case, then the expiration time should be equal to the time the password was changed plus the maximum reset age. The "expire without warning" option should not be taken into account for this case, because the intention of the maximum reset age is to ensure that the user changes his or her password in a timely manner after it has been reset by an administrator.

- When determining the expiration time relative to the forced change time (if one is defined), then it should only be considered if the forced change time is in the future, because if the forced change time has already arrived, then the account should be treated as if it were in a reset mode. The "expire without warning" option should not be taken into account for this case because this option will not actually cause the user's password to expire, but rather to enter a reset state.

Determining Whether a Proposed Password Is Acceptable

In order to determine whether a potential password is acceptable for use in a user's entry, a number of checks will be performed. The set of checks may be different depending on whether the password is being changed by the end user or reset by an administrator. The process that should be used to make this determination may follow the provided pseudocode:

```
if (requireSecurePasswordChanges && (! clientConnectionIsSecure))
{
  return false;
}

if (passwordIsPreEncoded)
{
  if (allowPreEncodedPasswords)
  {
    return true;
  }
  else
  {
    return false;
  }
}

if (changeRequestedByAdministrator)
{
  if (skipVaildationForAdministrators || passwordValidators.isEmpty())
  {
    return true;
  }

  for (PasswordValidator validator : passwordValidators)
  {
    if (! validator.passwordIsValid(newPassword))
    {
      return false;
    }
  }

  return true;
}

if (userAccountIsInactivated || userAccountIsLocked)
{
  return false;
}

if ((minAge > 0) && ((currentTime - passwordChangedTime) < minAge))
{
  return false;
```

```
}

if (userPasswordIsExpired && (! allowExpiredPasswordChanges))
{
  return false;
}

if (userPasswordIsReset && (maxResetAge > 0) &&
    ((currentTime - passwordChangedTime) > maxResetAge))
{
  return false;
}

if (requireCurrentPassword && (! validCurrentPasswordWasProvided))
{
  return false;
}

for (PasswordValidator validator : passwordValidators)
{
  if (! validator.passwordIsValid(newPassword))
  {
    return false;
  }
}

return true;
```

Simple Bind Request Processing

A simple bind request is used to authenticate to the Directory Server with a DN and password. There is a significant amount of processing that occurs when a simple bind is requested, and much of that work is not directly pertinent to password policy feature evaluation and therefore will be omitted from this document. However, there is a lot of work done that is directly related to password policy processing, and that process is described below. It is assumed that the user entry exists and is available for this processing.

1 The `pwdPolicySubentry` operational attribute is retrieved from the authenticating user's entry. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default password policy should be used. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the bind will fail.

2 A check is made to determine whether the user account has been disabled by an administrator. If the account has been inactivated, then the bind will fail.

3 A check is made to determine whether the authentication request must be processed in a secure manner. If so, and the underlying connection is not secure, then the bind will fail.

4 A check is made to determine whether the user account has been locked out due to too many failed attempts.
   4.1 If the user is currently locked out for this reason, and the lockout duration has not expired, then the bind will fail.
   4.2 If the user had previously been locked out but the lockout duration had passed, then the `pwdAccountLockedTime` attribute will be removed from the user's entry. The bind will not fail as a result of too many failed authentication attempts, although it may still fail for other reasons.

5 A check is made to determine whether the user's password has been administratively reset, and if so whether a maximum reset age will be enforced.
   5.1 If a maximum reset age will be enforced, and if that time period has elapsed, then the bind will fail.
   5.2 If the user's password has been reset by an administrator but the maximum reset age has not been reached, then a flag will be set to indicate that information about the account reset should be included in the response to the client. The bind will not fail as a result of this, although it may still fail for other reasons.

6 A check is made to determine whether a forced change time has been defined in the password policy.
   6.1 If there is no forced change time defined in the password policy, or if there is a forced change time but the user's password has already been changed in accordance

with it, then there will not be any impact on the success or failure of the bind request.

6.2 If there is a forced change time that is in the future, then it should be evaluated as a potential expiration time. If the current time plus the warning interval is greater than the forced change time, then a flag will be set to indicate that the user's password will expire in the near future. If this is the first time that the user has been warned of this, then the `ds-passwordWarnedTime` operational attribute should be set to the current time in the user's entry and an alert notification will be generated.

6.3 If there is a forced change time that is in the past, then the user's account will be treated as if it were in a reset mode. A flag will be set to include information about the reset in the response to the client.

7 A check is made to determine whether the user account has been locked out as a result of remaining idle for too long. If the user is locked out for this reason, then the bind will fail and an alert notification will be generated.

8 A check is made to determine whether the user's password is expired.

8.1 If the user's password is expired and the "expire without warning" configuration option is set to "`true`", and the user does not have any remaining grace logins, then the bind will fail and an alert notification will be generated.

8.2 If the user's password is expired and the "expire without warning" configuration option is set to "`true`", and the user has one or more grace logins remaining, then a flag will be set indicating that the `pwdGraceUseTime` operational attribute should be updated to include a value with the current time if the authentication is successful (a grace login should not be used if the authentication fails), and that a password expiration notification should be sent to the client.

8.3 If the user's password is expired, the "expire without warning" configuration option is undefined or set to "`false`", and the user has not previously been warned of the expiration, then this will be the first warning. The `ds-passwordWarnedTime` operational attribute will be set to the current time in the user's entry and an alert notification will be generated. The bind will not fail as a result of password expiration, although it may still fail for other reasons.

8.4 If the user's password is expired, the "expire without warning" configuration option is undefined or set to "`false`", and the current time is less than the warned time plus the warning interval, then a flag will be set to indicate that the user should receive an expiration warning. The bind will not fail as a result of password expiration, although it may still fail for other reasons.

8.5 If the user's password is expired, the "expire without warning" configuration option is undefined or set to "`false`", the current time is greater than or equal to than the warned time plus the warning interval, and the user does not have any remaining grace logins, then the bind will fail and an alert notification will be generated.

8.6 If the user's password is expired, the "expire without warning" configuration option is undefined or set to "`false`", the current time is greater than or equal to than the warned time plus the warning interval, and the user has one or more grace logins remaining, then a flag will be set indicating that the `pwdGraceUseTime` operational

attribute should be updated to include a value with the current time if the authentication is successful (a grace login should not be used if the authentication fails), and that a password expiration notification should be sent to the client.

9 A check is made to determine whether the password provided by the client is valid. If the provided password does match a stored password for the user, then the password should be considered valid. If the provided password does not match a stored password, then the `pwdFailureTime` attribute should be updated to indicate the most recent failure (checking to ensure that the new failure time does not duplicate an existing failure time, in the event that multiple failures include within the same period of granularity for the timestamp) and the bind will fail.

10 A check is made to determine whether the user's entry contains any password values that have been encoded using a deprecated storage scheme. If any such passwords exist, and the provided password matches the encoded representation for those passwords, then they will be removed and replaced with the same password encoded using the default storage scheme(s) for the password policy.

11 If both a last login time attribute and last login time format have been defined in the password policy configuration, then the Directory Server will generate a timestamp in the appropriate format. If that timestamp is different from any existing last login time value that may be present in the user's entry, then the entry should be updated with the new last login time (i.e., the update should not be attempted if the stored last login time matches the generated value).

It is important to point out that the order in which these checks will be performed has been intentionally designed so that the act of actually validating the password occurs as late as possible in the process. This can help to avoid the possibility of revealing any information that might help an attacker determine whether or not the provided password was valid if the authentication should have failed for any other reason.

It should also be noted that there will only be a single update to the user's entry as a result of password policy processing. If there are any modifications to perform, then they will be collected and applied as a single update before returning the result to the client. If there are one or more updates to be made to the user entry and a failure occurs while attempting to update that entry, then the authentication will fail.

Once all associated password policy processing has been completed, along with all other appropriate bind operation processing that may still remain, the response should be sent to the client indicating whether the authentication was successful. If the client included the password policy control in the bind request, then the corresponding response control will always be included in the reply, and it will include any appropriate components that may apply to the bind operation (but if it was requested, then the corresponding response control will still be provided even if the value sequence should be empty). If the password policy response control is included, then the Netscape password expiring or password expired controls will not be used. If the password policy response control is not included in the response and the user's password will expire in the near

future, then the Netscape password expiring control will be included in the response. If the password policy response control is not included in the response and the user's password is expired (regardless of whether a grace login has been used) or the account is in a reset mode, then the Netscape password expired control will be included in the response.

SASL Bind Request Processing

The SASL framework provides a great deal of flexibility in the area of authentication. However, it also introduces a significant degree of complexity with regard to password policy enforcement. There are many reasons for this added complexity, including:

- Some SASL mechanisms (e.g., `EXTERNAL` and `GSSAPI`) don't use a password at all.
- Some SASL mechanisms (e.g., `DIGEST-MD5` and `CRAM-MD5`) are password-based but the clear-text password is not included in the request.
- In general, SASL mechanisms do not use a bind DN but rather provide an authentication ID. This is not the same as a bind DN, and in these cases it is not possible for the core server to determine the identity of the user until the SASL processing is complete.
- The SASL framework is extensible, and it is possible to develop custom SASL mechanisms for use with the Directory Server. As a result, the requirements imposed on SASL mechanisms with regard to password policy processing should be minimal. Similarly, most password policy processing should be kept in the core bind operation code to avoid unnecessary duplication of the same processing in multiple SASL mechanism implementations.
- SASL authentications may require multiple stages of processing (e.g., `CRAM-MD5` and `DIGEST-MD5` each have two stages, and `GSSAPI` has three), and in such cases it will likely be the case that there will not be enough information to perform the necessary password policy evaluation until after the last stage is complete.

This means that from the standpoint of the password policy processing, the method of handling a SASL bind request will need to be different from the mechanism used for simple binds. The password policy processing will have to be done after all of the SASL mechanism processing for the core server to be able to determine the identity of the user that is attempting to authenticate. Despite this, the core server should still attempt to prevent disclosing any information about the validity of the credentials if there might be some other reason for the authentication to fail.

The general flow of operations for password policy processing on a SASL bind request is described below. Note that this will occur after all of the actual SASL processing for any request that does not require any further processing (i.e., any case in which the response is not going to be "SASL bind in progress").

1. The `pwdPolicySubentry` operational attribute is retrieved from the authenticating user's entry. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default password policy

should be enforced. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the bind will fail.

2 The `BindOperation.getSASLAuthUserEntry()` method is invoked to retrieve the entry for the user that is attempting to authenticate to the Directory Server. If this returns `null`, then either the processing was not sufficient to identify the user or there is no user associated with the authentication ID, and therefore no further password policy processing will be allowed. This will have no impact on the success or failure of the bind operation, because a `null` user entry does not necessarily mean that the authentication failed (e.g., it could have been an anonymous authentication).

3 A check is made to determine whether the user account has been disabled by an administrator. If the account has been inactivated, then the bind will fail.

4 A check is made to determine whether the authentication request must be processed in a secure manner. If so, and neither the underlying connection nor the SASL mechanism should be considered secure, then the bind will fail.

5 A check is made to determine whether the user account has been locked out due to too many failed attempts.
   5.1 If the user is currently locked out for this reason, and the lockout duration has not expired, then the bind will fail.
   5.2 If the user had previously been locked out but the lockout duration had passed, then the `pwdAccountLockedTime` attribute will be removed from the user's entry. The bind will not fail as a result of too many failed authentication attempts, although it may still fail for other reasons.

6 If the associated SASL mechanism is password-based, a check is made to determine whether the user's password has been administratively reset, and if so whether a maximum reset age will be enforced.
   6.1 If a maximum reset age will be enforced, and if that time period has elapsed, then the bind will fail.
   6.2 If the user's password has been reset by an administrator but the maximum reset age has not been reached, then a flag will be set to indicate that information about the account reset should be included in the response to the client. The bind will not fail as a result of this, although it may still fail for other reasons.

7 If the associated SASL mechanism is password-based, a check is made to determine whether a forced change time has been defined in the password policy.
   7.1 If there is no forced change time defined in the password policy, or if there is a forced change time but the user's password has already been changed in accordance with it, then there will not be any impact on the success or failure of the bind request.
   7.2 If there is a forced change time that is in the future, then it should be evaluated as a potential expiration time. If the current time plus the warning interval is greater than the forced change time, then a flag will be set to indicate that the user's

password will expire in the near future. If this is the first time that the user has been warned of this, then the `ds-passwordWarnedTime` operational attribute should be set to the current time in the user's entry and an alert notification will be generated.

7.3 If there is a forced change time that is in the past, then the user's account will be treated as if it were in a reset mode. A flag will be set to include information about the reset in the response to the client.

8 A check is made to determine whether the user account has been locked out as a result of remaining idle for too long. If the user is locked out for this reason, then the bind will fail and an alert notification will be generated.

9 If the associated SASL mechanism is password-based, a check is made to determine whether the user's password is expired.

9.1 If the user's password is expired and the "expire without warning" configuration option is set to "`true`", and the user does not have any remaining grace logins, then the bind will fail and an alert notification will be generated.

9.2 If the user's password is expired and the "expire without warning" configuration option is set to "`true`", and the user has one or more grace logins remaining, then a flag will be set indicating that the `pwdGraceUseTime` operational attribute should be updated to include a value with the current time if the authentication is successful (a grace login should not be used if the authentication fails), and that a password expiration notification should be sent to the client.

9.3 If the user's password is expired, the "expire without warning" configuration option is undefined or set to "`false`", and the user has not previously been warned of the expiration, then this will be the first warning. The `ds-passwordWarnedTime` operational attribute will be set to the current time in the user's entry and an alert notification will be generated. The bind will not fail as a result of password expiration, although it may still fail for other reasons.

9.4 If the user's password is expired, the "expire without warning" configuration option is undefined or set to "`false`", and the current time is less than the warned time plus the warning interval, then a flag will be set to indicate that the user should receive an expiration warning. The bind will not fail as a result of password expiration, although it may still fail for other reasons.

9.5 If the user's password is expired, the "expire without warning" configuration option is undefined or set to "`false`", the current time is greater than or equal to than the warned time plus the warning interval, and the user does not have any remaining grace logins, then the bind will fail and an alert notification will be generated.

9.6 If the user's password is expired, the "expire without warning" configuration option is undefined or set to "`false`", the current time is greater than or equal to than the warned time plus the warning interval, and the user has one or more grace logins remaining, then a flag will be set indicating that the `pwdGraceUseTime` operational attribute should be updated to include a value with the current time if the authentication is successful (a grace login should not be used if the authentication fails), and that a password expiration notification should be sent to the client.

10 If the SASL mechanism is password-based, then a check is made to determine whether the provided credentials were valid. If the credentials were not valid, then the `pwdFailureTime` attribute should be updated to indicate the most recent failure (checking to ensure that the new failure time does not duplicate an existing failure time, in the event that multiple failures include within the same period of granularity for the timestamp) and the bind will fail.

11 If both a last login time attribute and last login time format have been defined in the password policy configuration, then the Directory Server will generate a timestamp in the appropriate format. If that timestamp is different from any existing last login time value that may be present in the user's entry, then the entry should be updated with the new last login time (i.e., the update should not be attempted if the stored last login time matches the generated value).

As with simple authentication, there should be at most a single update to the user's entry as a result of this processing. If there are multiple changes that should be applied to it, then they should be batched into a single update.

Proxied Authorization Request Processing

The proxied authorization control may be used to request that an operation be performed under the authority of one user while bound as another. This control is commonly used in applications that are accessed by large numbers of users and maintain a pool of connections to the Directory Server (e.g., many directory-enabled Web-based applications). Strictly speaking, the use of this control does not constitute any authentication, but some password policy processing may still be required because if the user specified in the authorization ID is locked out, then attempts to proxy operations on behalf of that user should fail as well.

The following password policy processing should occur for any request that includes the proxied authorization control, just after retrieving the user entry specified as the authorization ID (unless the authorization ID indicated that the operation should be performed anonymously):

1 The `pwdPolicySubentry` operational attribute is retrieved from the proxied user's entry. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default password policy should be used. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the proxied operation will fail.

2 A check is made to determine whether the user account has been disabled by an administrator. If the account has been inactivated, then the proxied operation will fail.

3 A check is made to determine whether the user account has been locked out due to too many failed attempts.
   3.1 If the user is currently locked out for this reason, and the lockout duration has not expired, then the proxied operation will fail.
   3.2 If the user had previously been locked out but the lockout duration had passed, then the `pwdAccountLockedTime` attribute will be removed from the user's entry. The proxied operation will not fail as a result of too many failed authentication attempts, although it may still fail for other reasons.

4 A check is made to determine whether the user account has been locked out as a result of remaining idle for too long. If the user is locked out for this reason, then the proxied operation will fail and an alert notification will be generated.

5 A check is made to determine whether the user's password is expired. If the user's password is expired (including any potential warning notifications) and there are no grace logins available, then the proxied operation will fail and an alert notification will be

generated. The use of the proxied authorization control will not consume any grace logins or generate a password expiration warning notification.

This should be a streamlined set of processing, and in most cases will not require any updates to the proxied user's entry. Note that in some cases (e.g., a password is changed or reset using the proxied authorization control), some of the password policy processing may be duplicated. However, this should not introduce any significant performance penalty because any operations that may be duplicated would be very fast read-only checks.

Add Request Processing

The process for handling an add request is significantly different from that of handling a bind request, for many reasons. A bind operation is always considered to be performed by the end user, while it will be assumed that an add operation will always be performed by an administrator. Further, because the new entry does not exist, there is no need to determine whether the associated user may be inactivated or locked out. Finally, there is no additional penalty for updating the user's entry because it is going to be written to the database anyway, so it will be possible to piggyback any updates needed for password policy processing into that update.

The core set of processing that should be performed for an add operation is given below. This processing is performed after making sure that the entry is complete (e.g., after adding any missing RDN attributes or superior objectclasses) and just before the target entry is checked for schema compliance.

1. The `pwdPolicySubentry` operational attribute is retrieved from the entry being added. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default password policy should be used. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the add operation will fail.

2. A check is made to determine whether the provided entry contains a password. If it does not, then no further password policy processing will be performed for the add operation. This will have no impact on the success or failure of the add operation, as there will be many valid entries that do not contain a password.

3. If the `ds-cfg-require-secure-password-changes` configuration attribute has a value of "`true`" and the client connection used to request the add is not considered secure, then the add operation will fail.

4. If the provided password attribute contains any subtypes or other attribute options, then the entry will be rejected. The password attribute will not be allowed to have any attribute options.

5. If the password attribute has multiple values, then the `ds-cfg-allow-multiple-password-values` configuration attribute will be examined to determine whether multiple password values will be allowed. If not, then the add operation will fail.

6. Each password value will be examined to determine whether it is pre-encoded. If it is pre-encoded then the values for both the `ds-cfg-skip-validation-for-administrators` and the `ds-cfg-allow-pre-encoded-passwords` attributes will be retrieved. If validation should not be skipped for administrators and pre-encoded passwords will not be allowed, then the add operation will fail.

7 If the password value is not pre-encoded, the `ds-cfg-skip-validation-for-administrators` configuration attribute is absent or has a value of anything other than "`true`", and one or more password validators were defined for the password policy, then the provided plaintext password will be checked against those password validators. If any of the password validators rejects the provided password, then the add operation will fail.

8 If the password value is not pre-encoded, then the provided plaintext password will be encoded using all of the default storage schemes for the password policy.

9 The `pwdChangedTime` operational attribute will be added to the entry with a value equal to the current time represented in the generalized time syntax.

10 If the `ds-cfg-force-change-on-add` configuration attribute has a value of "`true`", then the `pwdReset` operational attribute will be added to the user entry with a value of "`true`" to indicate that the user will be required to change the password upon first authenticating to the server.

Modify Request Processing

Password policy processing for modify operations (as well as for the password modify extended operation) may be somewhat complicated because they may be performed either by the end user (i.e., a user changes his/her own password) or by an administrator (i.e., a user changes the password for another user). Some of the password policy processing that applies for a password change by an end user does not necessarily apply for administrative password changes, and vice versa. For this reason, the discussion of the password policy processing that needs to occur for a modify operation will be separated into two parts.

In order to determine whether the target operation is a user password change or an administrative password reset, the authorization DN for the modify request will be compared against the DN of the entry to modify. If the authorization DN is equal to the DN of the entry to be modified, then the operation will be considered a user password change. If the authorization DN is not equal to the DN of the entry to modify, then the operation will be considered an administrative password reset.

The password policy processing that should be performed for a modify by an end user includes:

1. The `pwdPolicySubentry` operational attribute is retrieved from the entry being modified. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default policy should be used. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the modify will fail.

2. A check is made to determine whether the changes in the provided modification will alter the password attribute in any way. If none of the changes impact the password attribute, then no further password policy processing will be performed for the modify operation. This will have no impact on the success or failure of the modify operation.

3. If the `ds-cfg-allow-user-password-changes` configuration attribute has a value of "`false`", then the modify operation will fail because users are not allowed to change their own passwords.

4. If the `ds-cfg-require-secure-password-changes` configuration attribute has a value of "`true`" and the client connection used to request the modify is not considered secure, then the modify operation will fail.

5. If the `ds-cfg-password-change-requires-current-password` configuration attribute has a value of "`true`", then the modify operation will fail, because user password changes in this mode will only be allowed through the use of the password modify extended operation.

6. If the updates to the password contain any subtypes or other attribute options, then the modify operation will fail. The password attribute will not be allowed to have any attribute options.

7. If the `ds-cfg-minimum-password-age` configuration attribute has a value greater than zero and the current time minus the time the password was last changed is less than the minimum allowed password age, then the modify operation will fail.

8. If the resulting update will cause the password attribute to have multiple different values (not to be confused with multiple encodings for the same password), then the `ds-cfg-allow-multiple-password-values` configuration attribute will be retrieved to determine whether multiple password values will be allowed. If not, then the modify operation will fail.

9. If modify operation will add any new password values in a pre-encoded form, then the `ds-cfg-allow-pre-encoded-passwords` attribute will be checked to determine whether this will be allowed. If pre-encoded passwords are not allowed, then the modify operation will fail.

10. If the modify operation will add any new password values in plaintext form, then they should be checked against any password validators configured for the password policy. If any of the password validators reject a password, then the modify operation will fail.

11. If the modify operation will add any new password values in plaintext form, then they should be encoded by the default set of password storage schemes configured for the password policy.

12. The `pwdChangedTime` operational attribute in the user's entry should be updated to contain a timestamp with the current time in the generalized time syntax.

13. The `pwdGraceUseTime`, `pwdFailureTime`, `pwdReset`, and `ds-pwp-warned-time` attributes should be removed from the user's entry if they exist.

14. If the `ds-cfg-require-change-by-time` configuration attribute exists in the password policy definition and has a value that is different from the value of the `ds-pwp-password-changed-by-required-time` operational attribute in the user's entry, then the user's entry should be updated so that the `ds-pwp-password-changed-by-required-time` has a value that matches that of the `ds-cfg-require-change-by-time` configuration attribute.

15. An account status notification message should be generated to indicate that the user's password has been changed.

The password policy processing that should be performed for a modify by an administrator includes:

1. The `pwdPolicySubentry` operational attribute is retrieved from the entry being modified. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default policy should be used. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the modify will fail.

2. A check is made to determine whether the changes in the provided modification will alter the password attribute in any way. If none of the changes impact the password attribute, then no further password policy processing will be performed for the modify operation. This will have no impact on the success or failure of the modify operation.

3. If the `ds-cfg-require-secure-password-changes` configuration attribute has a value of "`true`" and the client connection used to request the modify is not considered secure, then the modify operation will fail.

4. If the updates to the password contain any subtypes or other attribute options, then the modify operation will fail. The password attribute will not be allowed to have any attribute options.

5. If the resulting update will cause the password attribute to have multiple different values (not to be confused with multiple encodings for the same password), then the `ds-cfg-allow-multiple-password-values` configuration attribute will be retrieved to determine whether multiple password values will be allowed. If not, then the modify operation will fail.

6. If modify operation will add any new password values in a pre-encoded form, then the `ds-cfg-allow-pre-encoded-passwords` and `ds-cfg-skip-validation-for-administrators` configuration attributes will be checked to determine whether this will be allowed. If pre-encoded passwords are not allowed and administrative password changes are subject to validation, then the modify operation will fail.

7. If the modify operation will add any new password values in plaintext form and the `ds-cfg-skip-validation-for-administrators` configuration attribute does not have a value of "`true`", then they should be checked against any password validators configured for the password policy. If any of the password validators reject a password, then the modify operation will fail.

8. If the modify operation will add any new password values in plaintext form, then they should be encoded by the default set of password storage schemes configured for the password policy.

9. The `pwdChangedTime` operational attribute in the user's entry should be updated to contain a timestamp with the current time in the generalized time syntax.

10. If the `ds-cfg-force-change-on-reset` configuration attribute is set to "`true`", then the value of the `pwdReset` attribute in the user's entry will be set to "`true`".

11. The `pwdGraceUseTime`, `pwdFailureTime`, `pwdAccountLockedTime`, and `ds-pwp-warned-time` attributes should be removed from the user's entry if they exist.

12. If the `ds-cfg-require-change-by-time` configuration attribute exists in the password policy definition and has a value that is different from the value of the `ds-pwp-password-changed-by-required-time` operational attribute in the user's entry, then the user's entry should be updated so that the `ds-pwp-password-changed-by-required-time` has a value that matches that of the `ds-cfg-require-change-by-time` configuration attribute.

13. An account status notification message should be generated to indicate that the user's password has been reset by an administrator.

Note that as with the add operation, any changes that need to be made to the entry as a result of password policy processing should simply be merged into the overall set of changes to apply to the entry so that only a single update is required. If the modify operation fails for any reason, then no password policy updates will need to be made to the target entry.

Password Modify Request Processing

The use of the password modify extended operation is similar to simply performing a modify operation on the password attribute, but there are notable differences. For example, it is possible in either case to simply not provide a password and allow a password generator to create one. Similarly, this operation may be used under certain circumstances to allow a user to change an expired password even if that user cannot authenticate to the server.

For the standard modify operation, it was necessary to differentiate between user password changes and administrative password reset operations. For the password modify extended operation, it is actually necessary to further subdivide user password changes into two categories: those that are performed while authenticated as the end user versus those that are performed by an unauthenticated client but include the user's current password. In the second of these two cases, additional checks will be necessary to determine whether the user will be allowed to authenticate to the server before processing the password change.

Another important point that should be noted is that any case in which the target user's current password is provided as part of the password modify request (regardless of whether it is a user change or an administrative reset) must be treated in a security-conscious manner. In this case, no textual error message should be included in the response to the client, and any failure should cause the "unwilling to perform" result (result code 53) to be returned. This will help to ensure that the response to the client does not reveal any information about the validity of the client's password that might be useful to an attacker. Further, if the password policy request control was included, then the corresponding response control must ensure that the value sequence is empty.

For the case in which an authenticated user wishes to use the password modify extended operation to change his/her own password, the password policy processing should be performed as follows:

1. The `pwdPolicySubentry` operational attribute is retrieved from the entry being modified. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default policy should be used. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the password modify operation will fail.

2. If the `ds-cfg-allow-user-password-changes` configuration attribute has a value of "`false`", then the password modify operation will fail because users are not allowed to change their own passwords.

3. If the `ds-cfg-require-secure-password-changes` configuration attribute has a value of "`true`" and the client connection used to request the password modify operation is not considered secure, then the modify operation will fail.

4. If the user's current password is provided and the `ds-cfg-require-secure-authentication` attribute has a value of "`true`" and the client connection used to request the password modify operation is not considered secure, then the password modify operation will fail.

5. If the `ds-cfg-minimum-password-age` configuration attribute has a value greater than zero and the current time minus the time the password was last changed is less than the minimum allowed password age, then the password modify operation will fail.

6. If the password modify request does not include the user's current password and the `ds-cfg-password-change-requires-current-password` configuration attribute has a value of "`true`", then the password modify operation will fail.

7. If the password modify request includes the current password for the user (regardless of whether it is required by the `ds-cfg-password-change-requires-current-password` configuration attribute), then the Directory Server must verify that the current password is correct for that user. If the provided current password is incorrect, then the password modify operation will fail.

8. If the password modify request provides a new password and that password is pre-encoded, then the `ds-cfg-allow-pre-encoded-passwords` attribute will be checked to determine whether this will be allowed. If pre-encoded passwords are not allowed, then the password modify operation will fail.

9. If the password modify request provides a new plaintext password and one or more password validators are configured for the password policy, then that password should be checked against those password validators. If any of the password validators reject the new password, then the password modify operation will fail.

10. If the user's current password was provided, then all encoded passwords matching that current password will be removed from the user's entry. If the user's current password was not provided, then all existing password values will be removed from the user's entry.

11. If the password modify request provides a new plaintext password, then that password will be encoded by the default set of password storage schemes configured for the password policy.

12. If the password modify request does not provide a new password and there is no password validator configured for use with the password policy, then the password modify operation will fail. If a password validator is configured, then it will be used to generate a new password for the user and it will be encoded by the default set of password storage schemes configured for the password policy.

13. The `pwdChangedTime` operational attribute in the user's entry should be updated to contain a timestamp with the current time in the generalized time syntax.

14. The `pwdGraceUseTime`, `pwdFailureTime`, `pwdReset`, and `ds-pwp-warned-time` attributes should be removed from the user's entry if they exist.

15. If the `ds-cfg-require-change-by-time` configuration attribute exists in the password policy definition and has a value that is different from the value of the `ds-pwp-password-changed-by-required-time` operational attribute in the user's entry, then the user's entry should be updated so that the `ds-pwp-password-changed-by-required-time` has a value that matches that of the `ds-cfg-require-change-by-time` configuration attribute.

16. An account status notification message should be generated to indicate that the user's password has been changed.

For the case in which an unauthenticated user wishes to use the password modify extended operation to change a user password, and the current password for that user is included in the request, the password policy processing should be performed as follows:

1 The `pwdPolicySubentry` operational attribute is retrieved from the user's entry. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default password policy should be used. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the password modify operation will fail.

2 A check is made to determine whether the user account has been disabled by an administrator. If the account has been inactivated, then the password modify operation will fail.

3 A check is made to determine whether the authentication request must be processed in a secure manner. If so, and the underlying connection is not secure, then the password modify operation will fail.

4 A check is made to determine whether password changes must be processed in a secure manner. If so, and the underlying connection is not secure, then the password modify operation will fail.

5 A check is made to determine whether the user account has been locked out due to too many failed attempts.
   5.1 If the user is currently locked out for this reason, and the lockout duration has not expired, then the password modify operation will fail.
   5.2 If the user had previously been locked out but the lockout duration had passed, then the `pwdAccountLockedTime` attribute will be removed from the user's entry.

6 A check is made to determine whether the user's password has been administratively reset, and if so whether a maximum reset age will be enforced. If a maximum reset age will be enforced, and if that time period has elapsed, then the password modify operation will fail.

7 A check is made to determine whether the user account has been locked out as a result of remaining idle for too long. If the user is locked out for this reason, then the password modify operation will fail.

8 A check is made to determine whether the user's password is expired (taking into account the expire without warning option and grace logins). If the user's password is expired and the `ds-cfg-allow-expired-password-changes` configuration attribute is not set to "`true`", then the password modify operation will fail.

9 A check is made to verify that the current password is correct for that user. If the provided current password is incorrect, then the password modify operation will fail.

10 If both a last login time attribute and last login time format have been defined in the password policy configuration, then the Directory Server will generate a timestamp in the appropriate format. If that timestamp is different from any existing last login time value that may be present in the user's entry, then the entry should be updated with the new last login time (i.e., the update should not be attempted if the stored last login time matches the generated value).

11 If the `ds-cfg-allow-user-password-changes` configuration attribute has a value of "`false`", then the password modify operation will fail because users are not allowed to change their own passwords.

12 If the `ds-cfg-minimum-password-age` configuration attribute has a value greater than zero and the current time minus the time the password was last changed is less than the minimum allowed password age, then the password modify operation will fail.

13 If the password modify request provides a new password and that password is pre-encoded, then the `ds-cfg-allow-pre-encoded-passwords` attribute will be checked to determine whether this will be allowed. If pre-encoded passwords are not allowed, then the password modify operation will fail.

14 If the password modify request provides a new plaintext password and one or more password validators are configured for the password policy, then that password should be checked against those password validators. If any of the password validators reject the new password, then the password modify operation will fail.

15 All encoded passwords matching the provided current password will be removed from the user's entry.

16 If the password modify request provides a new plaintext password, then that password will be encoded by the default set of password storage schemes configured for the password policy.

17 If the password modify request does not provide a new password and there is no password validator configured for use with the password policy, then the password modify operation will fail. If a password validator is configured, then it will be used to generate a new password for the user and it will be encoded by the default set of password storage schemes configured for the password policy.

18 The `pwdChangedTime` operational attribute in the user's entry should be updated to contain a timestamp with the current time in the generalized time syntax.

19 The `pwdGraceUseTime`, `pwdFailureTime`, `pwdReset`, and `ds-pwp-warned-time` attributes should be removed from the user's entry if they exist.

20 If the `ds-cfg-require-change-by-time` configuration attribute exists in the password policy definition and has a value that is different from the value of the `ds-pwp-password-changed-by-required-time` operational attribute in the user's entry, then the user's entry should be updated so that the `ds-pwp-password-changed-by-required-time` has a value that matches that of the `ds-cfg-require-change-by-time` configuration attribute.

21 An account status notification message should be generated to indicate that the user's password has been changed.

For the case in which an authenticated user wishes to use the password modify extended operation to change the password for another user, the password policy processing should be performed as follows:

1. The `pwdPolicySubentry` operational attribute is retrieved from the entry being modified. If this attribute exists (either explicitly stored in the entry or provided as a virtual attribute), then its value should be a DN that refers to the password policy that should be enforced. If the attribute does not exist, then the server-wide default policy should be used. If the attribute exists but does not contain the DN of the configuration entry for a valid password policy definition, then the password modify operation will fail.

2. If the `ds-cfg-require-secure-password-changes` configuration attribute has a value of "`true`" and the client connection used to request the modify is not considered secure, then the password modify operation will fail.

3. If the user's current password is provided and the `ds-cfg-require-secure-authentication` attribute has a value of "`true`" and the client connection used to request the password modify operation is not considered secure, then the password modify operation will fail.

4. If the password modify request provides a new password and that password is pre-encoded, then the `ds-cfg-allow-pre-encoded-passwords` and `ds-cfg-skip-validation-for-administrators` attributes will be checked to determine whether this will be allowed. If pre-encoded passwords are not allowed and administrative password changes are subject to validation, then the password modify operation will fail.

5. If the password modify request provides a new password in plaintext form and the `ds-cfg-skip-validation-for-administrators` configuration attribute does not have a value of "`true`", then they should be checked against any password validators configured for the password policy. If any of the password validators reject a password, then the password modify operation will fail.

6. If the user's current password was provided, then all encoded passwords matching that current password will be removed from the user's entry. If the user's current password was not provided, then all existing password values will be removed from the user's entry.

7. If the password modify request provides a new plaintext password, then that password will be encoded by the default set of password storage schemes configured for the password policy.

8. If the password modify request does not provide a new password and there is no password validator configured for use with the password policy, then the password modify operation will fail. If a password validator is configured, then it will be used to generate a new password for the user and it will be encoded by the default set of password storage schemes configured for the password policy.

9. The `pwdChangedTime` operational attribute in the user's entry should be updated to contain a timestamp with the current time in the generalized time syntax.

10. If the `ds-cfg-force-change-on-reset` configuration attribute is set to "`true`", then the value of the `pwdReset` attribute in the user's entry will be set to "`true`".

11. The `pwdGraceUseTime`, `pwdFailureTime`, `pwdAccountLockedTime`, and `ds-pwp-warned-time` attributes should be removed from the user's entry if they exist.

12. If the `ds-cfg-require-change-by-time` configuration attribute exists in the password policy definition and has a value that is different from the value of the `ds-pwp-password-changed-by-required-time` operational attribute in the user's entry, then the user's entry should be updated so that the `ds-pwp-password-changed-by-required-time` has a value that matches that of the `ds-cfg-require-change-by-time` configuration attribute.

13. An account status notification message should be generated to indicate that the user's password has been reset by an administrator.

LDIF Import Processing

During LDIF import processing, the password policy processing that the Directory Server must perform is minimal. The primary task that must be performed is to encode any plaintext passwords using the default password storage schemes for the associated password policy. This can be a somewhat complex task because it may not be straightforward to determine which password policy should be used. The primary issues that could arise in this case are that different password policies could use different password attributes and/or different default storage schemes for the password attributes.

For cases in which an entry to import contains a value for the `pwdPolicySubentry` then the associated password policy will be used. However, for entries that do not contain an explicit value for the `pwdPolicySubentry` operational attribute, it may be very difficult to determine whether it should use the default password policy or if an alternate policy should be used through a virtual attribute.

One possible approach to solving this problem would be to attempt to build the virtual attribute logic into the import process. However, this would degrade import performance to a degree, and it may be difficult or even impossible to accomplish depending on the data layout (e.g., if some of the information needed to construct the virtual attribute is in another backend whose contents are not available during the import).

Rather than attempting to perform all virtual attribute processing, a simpler and more efficient approach will be to examine all of the password policy definitions contained in the configuration and aggregate all default storage schemes for a given password attribute. For example, if one password policy uses a default storage scheme of "`SSHA`" for the `userPassword` attribute and a second password policy uses a default storage scheme of "`MD5`" for the `userPassword` attribute, then any `userPassword` values contained in the LDIF entries will be encoded using both the `SSHA` and `MD5` schemes.

In general, this should not be a problem. Most directories will not have a need to have different password policies with conflicting default storage schemes for the same attribute that would be selected using virtual attributes, and even for cases in which there are different sets of default storage schemes, there would rarely be a case in which having multiple encodings for those attributes would cause any significant problem. If there is a significant concern that this could cause a problem, then additional approaches could be investigated in the future.

Remaining Implementation Details

This document does not make any attempt to describe ways in which a virtual attribute subsystem may provide password policy state attributes like `pwdPolicySubentry` or `ds-pwp-account-disabled`. The virtual attribute mechanism should be described in a separate document.

Another significant area that is not addressed in this document is the interaction between a synchronization environment and the password policy. There will be a requirement to ensure that events like account lockout will be propagated quickly to other servers in the environment, or at least that requests will consistently be routed to the same server. More investigation into the interaction between the synchronization and/or routing subsystems must be delayed until additional work is done in those areas.

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.

1.12. "Source Code" means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of:

(1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients' rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient's rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY

COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as "Participant") alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or

provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction's conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

For Covered Software in this distribution, this License shall be governed by the laws of the State of California (excluding conflict-of-law provisions).

Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.