Network Working Group
INTERNET-DRAFT
Intended Category: Standards Track

Rosanna Lee
Sun Microsystems
Rob Weltman
Coscend Corp.
May, 2001

The Java SASL Application Program Interface
draft-weltman-java-sasl-05.txt

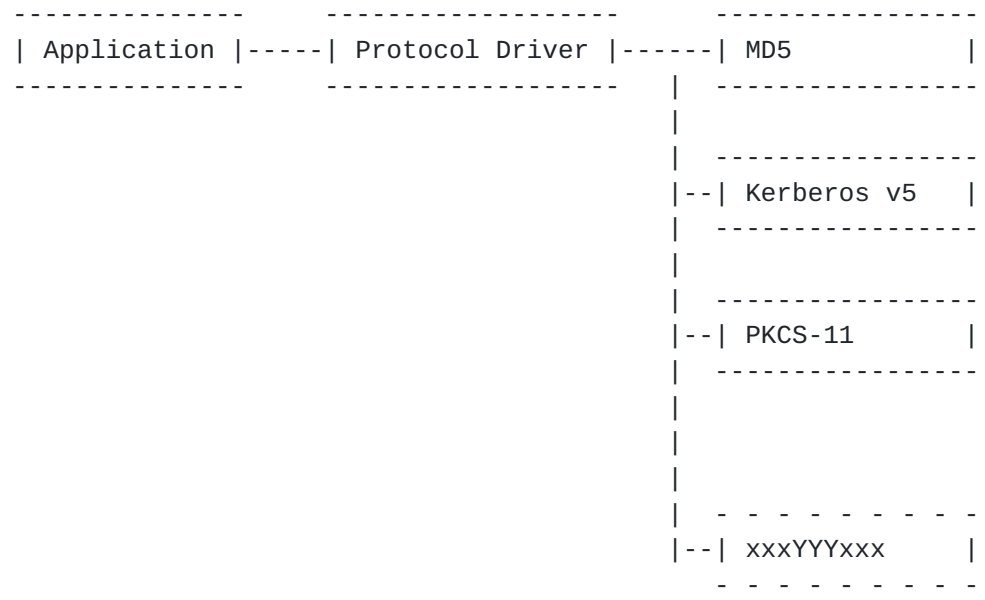
Status of this Memo

Abstract

This document defines a client-side and a server-side Java language interface for using the Simple Authentication and Security Layer (SASL) mechanisms for adding authentication support to connection-based protocols. The interface promotes sharing of SASL Mechanism Drivers and security layers between applications using different protocols. It complements but does not replace [SASL], which defines and exemplifies use of the SASL protocol in a language-independent way.

1. Introduction

See [SASL], section 3, for an introduction to and overview of the SASL framework for authentication and negotiation of a security layer. The following presents an outline of the concepts.



An application chooses a Protocol Driver specific to the protocol it wants to use, and specifies one or more acceptable mechanisms. The Protocol Driver controls the socket, and knows the format/packaging of bytes sent down and received from the socket, but does not know how to authenticate or to encrypt/ decrypt the bytes. It uses one of the Mechanism Drivers to help it perform authentication. The Protocol Driver examines each byte string received from the server during the authentication in a protocol-specific way to determine if the authentication process has been completed. If not, the byte string is passed to the Mechanism Driver to be interpreted as a server challenge; the Mechanism Driver returns an appropriate response, which the Protocol Driver can encode in a protocol-specific way and return to the server.

If the Protocol Driver concludes from the byte string received from the server that authentication is complete, it may query the Mechanism Driver if it considers the authentication process complete, in order to thwart early completion messages inserted by an intruder.

On completed authentication, the Protocol Driver may use the Mechanism Driver to encode and decode any data exchanged through the socket if a security layer was negotiated.

A complication here is that some authentication methods may require additional user/application input. That means that a Mechanism Driver may need to call up to an application during the authentication process. To satisfy this requirement, the application can supply a javax.security.auth.callback.CallbackHandler instance

[JAAS] that can be used by the Mechanism Driver to prompt the user for additional input.

Protocol Drivers are protocol-dependent and may be built in to a protocol package or an application. There is a generalized framework for registering and finding Mechanism Drivers. The framework uses a factory to produce an appropriate Mechanism Driver. The factory may be preconfigured, explicitly specified by the caller, specified as a list of packages by the caller, or be identified based on a list of packages in the System properties.

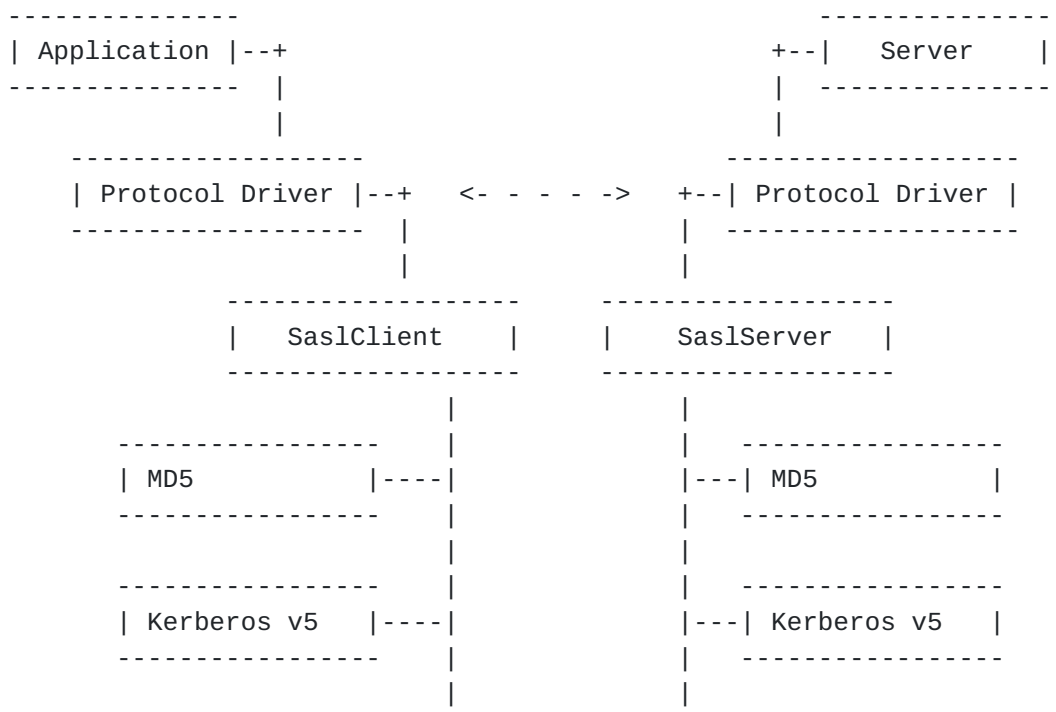
The Mechanism Drivers are protocol-independent, and don't deal directly with network connections, just byte arrays, so they can be implemented in a generalizable way for all protocols.

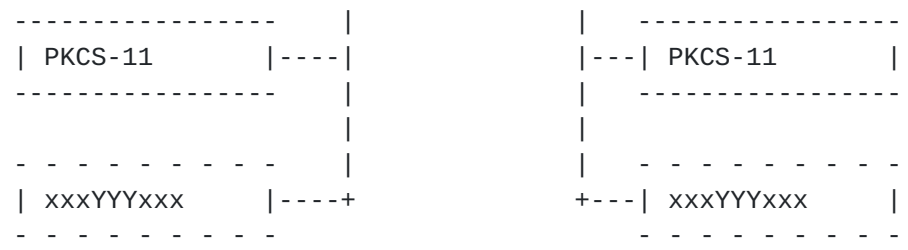
The negotiated security layer is implemented by encoding and decoding routines in the Mechanism Driver, using parameters and resolutions reached during authentication.

Different Mechanism Drivers may require different parameters to carry out the authentication process. This is handled by passing a java.util.Hashtable object as an argument to instantiation methods.

In the following discussion, 'client' refers to the client-side Protocol Driver that is using the SASL mechanism while 'server' refers to the server-side Protocol Driver that is using the SASL mechanism.

In the Java SASL environment, the SaslClient interface represents the client's view of the Mechanism Driver, while the SaslServer interface represents the server's view.





A client using the Java SASL API may communicate with any server implementing the SASL protocol, and a server may use the API to process authentication requests from any client using the SASL protocol. It is not required that both sides use the same language bindings.

2. Overview of the SASL classes

2.1 Interfaces

SaslClient	Performs SASL authentication as a client.
SaslClientFactory	An interface for creating instances of SaslClient. It is not normally accessed directly by a client, which will use the Sasl static methods instead. However, a particular environment may provide and install a new or different SaslClientFactory.
SaslServer	Performs SASL authentication as a server.
SaslServerFactory	An interface for creating instances of SaslServer. It is not normally accessed directly by a server, which will use the Sasl static methods instead. However, a particular environment may provide and install a new or different SaslServerFactory.

2.2 Classes

Sasl	A static class for creating SASL clients and servers. It transparently locates and uses any available SaslClientFactory/SaslServerFactory instances.
AuthorizeCallback	This callback is used by SaslServer to determine whether one entity (identified by an authenticated authentication id) can act on behalf of another entity (identified by an authorization id).
RealmCallback	This callback is used by SaslClient and SaslServer to retrieve realm information.
RealmChoiceCallback	This callback is used by SaslClient and SaslServer to obtain one or more realms given a list of realm choices.
SaslException	Exception thrown on errors and failures in the authentication process.

3. Overview of SASL API Use

An application generally uses the SASL API as follows:

- Create an object implementing the client authentication callback interfaces, which can provide credentials when required by the SaslClient.

- Pass a list of acceptable or known Mechanisms and a callback handler to Sasl.createSaslClient. The method returns an object implementing SaslClient on success.

- Have the SaslClient object begin the authentication process by providing an initial server response, if the protocol supports an initial response.

- Responses/challenges are exchanged with the server. If a response indicates authentication has completed, SaslClient is queried for validation, and methods for encoding and decoding data according to the negotiated security layer may be invoked on it. If not, the SaslClient is queried for an appropriate next response to the server. This continues until

authentication has completed.

- If a security layer has been negotiated, for the rest of the session, messages to the server are first encoded by using SaslClient before being written, and messages from the server are first decoded by using SaslClient before being processed in the application.

A server generally uses the SASL API as follows:

- It receives a request from the client requesting authentication for a particular SASL mechanism, accompanied by an optional initial response.

- It processes the initial response and generates a challenge specific to the SASL mechanism to be sent back to the client if the response is processed successfully. If the response is not processed successfully, it sends an error to the client and terminates the authentication session.

- Responses/challenges are exchanged with the client. If the server cannot successfully process a response, the server sends an error to the client and terminates the authentication. If the server has completed the authentication and has no more challenges to send, it sends a success indication to the client.

- If the authentication has completed successfully, the server extracts the authorization ID of the client from the SaslServer instance (if appropriate) to be used for subsequent access control checks.

- For the rest of the session, messages to and from the client are encoded and decoded by using SaslServer to implement the negotiated security layer (if any).

The following sections describe the SASL classes in more detail.

4. The Java SASL classes

4.1 public class Sasl

A class capable of providing a SaslClient or SaslServer.

4.1.1 createSaslClient

```
public static SaslClient
createSaslClient(String[] mechanisms,
                 String authorizationID,
                 String protocol,
                 String serverName,
                 Hashtable props,
                 javax.security.auth.callback.CallbackHandler cbh)
                 throws SaslException
```

Creates a SaslClient using the parameters supplied. It returns null if no SaslClient can be created using the parameters supplied. Throws SaslException if it cannot create a SaslClient because of an error.

The algorithm for selection is as follows:

1. If a factory has been installed via setSaslClientFactory(), invoke createSaslClient() on it. If the method invocation returns a non-null SaslClient instance, return the SaslClient instance; otherwise continue.
2. Create a list of fully qualified class names using the package names listed in the CLIENT_PKGS ("javax.security.sasl.client.pkgs") property in props and the class name ClientFactory. Each class name in this list identifies a SaslClientFactory implementation. Starting with the first class on the list, create an instance of SaslClientFactory using the class' public no-argument constructor and invoke createSaslClient() on it. If the method invocation returns a non-null SaslClient instance, return it; otherwise repeat using the next class on the list until a non-null SaslClient is produced or the list is exhausted.
3. Repeat the previous step using the CLIENT_PKGS ("javax.security.sasl.client.pkgs") System property instead of the property in props.
4. As per the Java 2 Standard Edition version 1.3 service provider guidelines, check for the existence of one of more files named META-INF/services/javax.security.sasl.SaslClientFactory in the classpath and installed JAR files. Each file lists the fully qualified class names of the factories (i.e. implementations of SaslClientFactory) found in the JAR files or classpath. Construct a merged list of class names using these files and repeat Step 2 using this list. If there are more than one of these files, the

order in which they are processed is undefined. If no non-null SaslClient instance is produced, return null.

Parameters are:

mechanisms	The non-null list of mechanism names to try. Each is the IANA-registered name of a SASL mechanism (e.g. "GSSAPI", "CRAM-MD5").
authorizationID	The possibly null protocol-dependent identification to be used for authorization. If null or empty, the server derives an authorization ID from the client's authentication credentials. When the SASL authentication completes successfully, the specified entity is granted access.
protocol	The non-null string name of the protocol for which the authentication is being performed, e.g "pop", "ldap".
serverName	The non-null fully qualified host name of the server to authenticate to.
props	The possibly null set of properties used to select the SASL mechanism and to configure the authentication exchange of the selected mechanism. For example, if props includes the Sasl.POLICY_NOPLAINTEXT property with the value "true", then the selected SASL mechanism must not be susceptible to simple plain passive attacks. In addition to the standard properties of this class, other, possibly mechanism-specific, properties can be included. Properties not relevant to the selected mechanism are ignored. See Standard Properties for a list of standard properties.
cbh	The possibly null callback handler to be used by the SASL mechanisms to get further information from the application/library to complete the authentication. For example, a SASL mechanism might require the authentication ID, password and realm from the caller. The authentication ID is requested by using a NameCallback. The password is requested by using a PasswordCallback. The realm is requested by using a RealmChoiceCallback if there is a list of realms to choose from, and

by using a RealmCallback if the realm must be entered.

4.1.2 setSaslClientFactory

```
public static void
setSaslClientFactory(SaslClientFactory fac)
```

Sets the default SaslClientFactory to use. This method sets fac to be the default factory. It can only be called with a non-null value once per VM. If a factory has been set already, this method throws IllegalStateException. The method throws java.lang.SecurityException if the caller does not have the necessary permission to set the factory.

Parameters are:

fac	The possibly null factory to set. If null, it doesn't do anything.

4.1.3 createSaslServer

```
public static SaslServer
createSaslServer(String mechanism,
                 String protocol,
                 String serverName,
                 Hashtable props,
                 javax.security.auth.callback.CallbackHandler cbh)
                 throws SaslException
```

This method creates a SaslServer for the specified mechanism. It returns null if no SaslServer can be created for the specified mechanism.

The algorithm for selection is as follows:

1. If a factory has been installed via setSaslServerFactory(), invoke createSaslServer() on it. If the method invocation returns a non-null SaslServer instance, return the SaslServer instance; otherwise continue.
2. Create a list of fully qualified class names using the package names listed in the SERVER_PKGS ("javax.security.sasl.server.pkgs") property in props and the class name ServerFactory. Each class name in this list identifies a SaslServerFactory implementation. Starting with the first class

on the list, create an instance of SaslServerFactory using the class' public no-argument constructor and invoke createSaslServer() on it. If the method invocation returns a non-null SaslServer instance, return it; otherwise repeat using the next class on the list until a non-null SaslServer is produced or the list is exhausted.

3. Repeat the previous step using the SERVER_PKGS ("javax.security.sasl.server.pkgs") System property instead of the property in props.

4. As per the Java 2 Standard Edition version 1.3 service provider guidelines, check for the existence of one of more files named META-INF/services/javax.security.sasl.SaslServerFactory in the classpath and installed JAR files. Each file lists the fully qualified class names of the factories (i.e. implementations of SaslServerFactory) found in the JAR files or classpath. Construct a merged list of class names using these files and repeat Step 2 using this list. If there are more than one of these files, the order in which they are processed is undefined. If no non-null SaslServer instance is produced, return null.

Parameters are:

mechanism	A non-null IANA-registered name of a SASL mechanism (e.g. "GSSAPI", "CRAM-MD5").
protocol	The non-null string name of the protocol for which the authentication is being performed, e.g "pop", "ldap".
serverName	The non-null fully qualified host name of the server.
props	The possibly null set of properties used to select the SASL mechanism and to configure the authentication exchange of the selected mechanism. For example, if props includes the Sasl.POLICY_NOPLAINTEXT property with the value "true", then the selected SASL mechanism must not be susceptible to simple plain passive attacks. In addition to the standard properties defined in this class, other, possibly mechanism-specific, properties can be included. Properties not relevant to the selected mechanism are ignored. See section 4.1.7 Standard Properties for a list

of standard properties.

cbh — The possibly null callback handler to be used by the SASL mechanism to get further information from the application/library to complete the authentication. For example, a SASL mechanism might require the authentication ID and password from the caller. The authentication ID is requested with a NameCallback, and the password with a PasswordCallback.

4.1.4 setSaslServerFactory

public static void

setSaslServerFactory(SaslServerFactory fac)

Sets the default SaslServerFactory to use. This method sets fac to be the default factory. It can only be called with a non-null value once per VM. If a factory has been set already, this method throws IllegalStateException. The method throws java.lang.SecurityException if the caller does not have the necessary permission to set the factory.

Parameters are:

fac — The possibly null factory to set. If null, it doesn't do anything.

4.1.5 getSaslClientFactories

public java.util.Enumeration
getSaslClientFactories(java.util.Hashtable props)

Gets an enumeration of known factories for producing SaslClient. This method uses the same sources for locating factories as createSaslClient().

Parameters are:

props — A possibly null set of properties that may contain policy properties and the property CLIENT_PKGS ("javax.security.sasl.client.pkgs") for specifying a list of SaslClientFactory implementation package names.

4.1.6 getSaslServerFactories

```
public java.util.Enumeration
getSaslServerFactories(java.util.Hashtable props)
```

Gets an enumeration of known factories for producing SaslServer. This method uses the same sources for locating factories as createSaslServer().

Parameters are:

props	A possibly null set of properties that may contain policy properties and the property SERVER_PKGS ("javax.security.sasl.server.pkgs") for specifying a list of SaslServerFactory implementation package names.

4.1.7 Standard Properties

There are a number of properties that may be specified in a Hashtable parameter when creating a SASL client or server. The standard properties and descriptions of their values are as follows (with the Sasl constant name followed by the literal value in parentheses):

QOP ("javax.security.sasl.qop")

A comma-separated, ordered list of quality-of-protection values that the client or server is willing to support. A qop value is one of

"auth"	authentication only
"auth-int"	authentication plus integrity protection
"auth-conf"	authentication plus integrity and confidentiality protection

The order of the list specifies the preference order of the client or server. If this property is absent, the default qop is "auth".

STRENGTH ("javax.security.sasl.strength")

A comma-separated, ordered list of cipher strength values that the client or server is willing to support. A strength value is one of

"low"

"medium"

"high"

The order of the list specifies the preference order of the client or server. An implementation SHOULD allow configuration of the meaning of these values.

An application MAY use the Java Cryptography Extension (JCE) with JCE-aware mechanisms to control the selection of cipher suites that match the strength values.

If this property is absent, the default strength is "high,medium,low".

SERVER_AUTH ("javax.security.sasl.server.authentication")

"true" if server must authenticate to client; default "false"

MAX_BUFFER ("javax.security.sasl.maxbuffer")

Maximum size of receive buffer in bytes of SaslClient/SaslServer; the default is defined by the mechanism. The property value is the string representation of an integer.

CLIENT_PKGS ("javax.security.sasl.client.pkgs")

A |-separated list of package names to use when locating a SaslClientFactory. Each package MUST contain a class named ClientFactory that implements the SaslClientFactory interface.

SERVER_PKGS ("javax.security.sasl.server.pkgs")

A |-separated list of package names to use when locating a SaslServerFactory. Each package MUST contain a class named ServerFactory that implements the SaslServerFactory

interface.

RAW_SEND_SIZE ("javax.security.sasl.rawsendsize")

Maximum size of the raw send buffer in bytes of SaslClient/SaslServer. The property value is the string representation of an integer and is negotiated between the client and server during the authentication exchange.

The following properties are for defining a security policy for a server or client. Absence of the property is interpreted as "false".

POLICY_NOPLAINTEXT ("javax.security.sasl.policy.noplaintext")

"true"	if mechanisms susceptible to simple plain passive attacks (e.g. "PLAIN") are not permitted
"false"	if such mechanisms are permitted

POLICY_NOACTIVE ("javax.security.sasl.policy.noactive")

"true"	if mechanisms susceptible to active (non-dictionary) attacks are not permitted
"false"	if such mechanisms are permitted.

POLICY_NODICTIONARY ("javax.security.sasl.policy.nodictionary")

"true"	if mechanisms susceptible to passive dictionary attacks are not permitted

"false"	if such mechanisms are permitted

POLICY_NOANONYMOUS ("javax.security.sasl.policy.noanonymous")

"true"	if mechanisms that accept anonymous login are not permitted
"false"	if such mechanisms are permitted

POLICY_FORWARD_SECRECY ("javax.security.sasl.policy.forward")

Forward secrecy means that breaking into one session will not automatically provide information for breaking into future sessions.

"true" if mechanisms that implement forward secrecy between sessions are required

"false" if such mechanisms are not required

POLICY_PASS_CREDENTIALS ("javax.security.sasl.policy.credentials")

"true" if mechanisms that pass client credentials are required

"false" if such mechanisms are not required

4.2 public interface SaslClient

An object implementing this interface can negotiate authentication as a client using one of the IANA-registered mechanisms.

4.2.1 evaluateChallenge

```
public byte[]
evaluateChallenge(byte[] challenge)
                  throws SaslException
```

If a challenge is received from the server during the authentication process, this method is called to prepare an appropriate next response to submit to the server. The response is null if the challenge accompanied a "SUCCESS" status and the challenge only contains data for the client to update its state and no response needs to be sent to the server. The response is a zero-length byte array if the client is to send a response with no data. A SaslException is thrown if an error occurred while processing the challenge or generating a response.

Parameters are:

challenge The non-null challenge received from the server. The challenge array may have zero length.

4.2.2 hasInitialResponse

```
public boolean hasInitialResponse()
```

Determines whether this mechanism has an optional initial response. If true, caller should call evaluateChallenge() with an empty array to get the initial response.

4.2.3 isComplete

```
public boolean
isComplete()
```

This method may be called at any time to determine if the authentication process is finished. Typically, the Protocol Driver will not do this until it has received indication from the server (in a protocol-specific manner) that the process has completed.

4.2.4 unwrap

```
public byte[] unwrap(byte[] incoming, int offset, int len )
                     throws SaslException
```

Unwraps a byte array received from the server to return the corresponding decoded bytes in a byte array.

This method can be called only after the authentication process has completed (i.e., when isComplete() returns true) and only if the authentication process has negotiated integrity and/or privacy as the quality of protection; otherwise, a SaslException is thrown. A SaslException is thrown also if incoming cannot be successfully unwrapped.

incoming is the contents of the SASL buffer as defined in [SASL] without the leading four octet field that represents the length. offset and len specify the portion of incoming to use.

Parameters are:

incoming	A non-null byte array containing the encoded bytes from the server.
offset	The starting position at incoming of the bytes to use.

len	The number of bytes from incoming to use.

4.2.5 wrap

```
public byte[] wrap(byte[] outgoing, int offset, int len)
                  throws SaslException
```

Wraps a byte array to be sent to the server to return the corresponding encoded bytes in a byte array.

This method can be called only after the authentication exchange has completed (i.e., when isComplete() returns true) and only if the authentication exchange has negotiated integrity and/or privacy as the quality of protection; otherwise, a SaslException is thrown. A SaslException is thrown also if outgoing cannot be successfully wrapped.

The result of this method will make up the contents of the SASL buffer as defined in [SASL] without the leading four octet field that represents the length.

offset and len specify the portion of outgoing to use.

Parameters are:

outgoing	A non-null byte array containing the bytes to encode.
offset	The starting position at outgoing of the bytes to use.
len	The number of bytes from outgoing to use.

4.2.6 getMechanismName

```
public String
getMechanismName()
```

Reports the IANA-registered name of the mechanism used by this client, e.g. "GSSAPI" or "CRAM-MD5".

4.2.7 getNegotiatedProperty

```
public String getNegotiatedProperty(String propName)
                                    throws SaslException
```

Retrieves the negotiated property. This method can be called only after the authentication exchange has completed (i.e., when isComplete() returns true); otherwise, a SaslException is thrown.

For example, this method may be used to obtained the negotiated raw send buffer size, quality-of-protection, and cipher strength. See Section 4.1.7 for a list of standard properties.

This method returns null when the specified property was not negotiated or is not applicable to this mechanism.

Parameters:

propName The non-null property name.

4.2.8 dispose

```
public abstract void dispose() throws SaslException
```

Disposes of any system resources or security-sensitive information the SaslClient might be using. Invoking this method invalidates the SaslClient instance. This method is idempotent.

4.3 public interface SaslClientFactory

An object implementing this interface can provide a SaslClient. The implementation must be thread-safe and handle multiple simultaneous requests. It must also have a public constructor that accepts no argument.

4.3.1 createSaslClient

```
public SaslClient
createSaslClient(String[] mechanisms,
                 String authorizationID,
                 String protocol,
                 String serverName,
                 Hashtable props,
                 javax.security.auth.callback.CallbackHandler cbh)
                 throws SaslException
```

Creates a SaslClient using the parameters supplied. It returns null if no SaslClient can be created using the parameters supplied. Throws SaslException if it cannot create a SaslClient because of an error.

Parameters are:

mechanisms The non-null list of mechanism names to try. Each is the IANA-registered name of a SASL mechanism (e.g. "GSSAPI", "CRAM-MD5").

authorizationID The possibly null protocol-dependent identification to be used for authorization. If null or empty, the server derives an authorization ID from the client's authentication credentials. When the SASL authentication completes successfully, the specified entity is granted access.

protocol The non-null string name of the protocol for which the authentication is being performed, e.g "pop", "ldap".

serverName The non-null fully qualified host name of the server to authenticate to.

props The possibly null set of properties used to select the SASL mechanism and to configure the authentication exchange of the selected mechanism. See the Sasl class for a list of standard properties. Other, possibly mechanism-specific, properties can be included. Properties not relevant to the selected mechanism are ignored.

cbh The possibly null callback handler to be used by the SASL mechanisms to get further information from the application/library to complete the authentication. For example, a SASL mechanism might require the authentication ID, password and realm from the caller. The authentication ID is requested by using a NameCallback. The password is requested by using a PasswordCallback. The realm is requested by using a RealmChoiceCallback if there is a list of realms to choose from, and by using a RealmCallback if the realm must be entered.

4.3.2 getMechanismNames

```
public String[]
```

getMechanismNames(Hashtable props)

Returns a non-null array of names of mechanisms supported by this factory that match the specified mechanism selection policies.

Parameters are:

props	The possibly null set of properties used to specify the security policy of the SASL mechanisms. For example, if props contains the Sasl.POLICY_NOPLAINTEXT property with the value "true", then the factory must not return any SASL mechanisms that are susceptible to simple plain passive attacks. See the Sasl class for a complete list of policy properties. Non-policy related properties, if present in props, are ignored.

4.4 public interface SaslServer

An object implementing this interface can negotiate authentication as a server using one of the IANA-registered mechanisms.

4.4.1 evaluateResponse

```
public byte[]
evaluateResponse(byte[] response)
                 throws SaslException
```

If a response is received from the client during the authentication process, this method is called to prepare an appropriate next challenge to submit to the client. The challenge is null if the authentication has succeeded and no more challenge data is to be sent to the client. It is non-null if the authentication must be continued by sending a challenge to the client, or if the authentication has succeeded but challenge data needs to be processed by the client. A SaslException is thrown if an error occurred while processing the response or generating a challenge. isComplete() should be called after each call to evaluateResponse() to determine if any further response is needed from the client. The Protocol Driver will send an indication (in a protocol-specific manner) as to whether the authentication has succeeded, failed, or should be continued, and any accompanying challenge data.

Parameters are:

response	Non-null response received from client.

4.4.2 isComplete

```
public boolean
isComplete()
```

This method may be called at any time to determine if the authentication process is finished. This method is typically called

after each invocation of evaluateResponse() to determine whether the authentication has completed successfully or should be continued.

4.4.3 unwrap

```
public byte[] unwrap(byte[] incoming, int offset, int len)
                     throws SaslException
```

Unwraps a byte array received from the client to return the corresponding decoded bytes in a byte array.

This method can be called only after the authentication process has completed (i.e., when isComplete() returns true) and only if the authentication process has negotiated integrity and/or privacy as the quality of protection; otherwise, a SaslException is thrown. A SaslException is thrown also if incoming cannot be successfully unwrapped.

incoming is the contents of the SASL buffer as defined in [SASL] without the leading four octet field that represents the length. offset and len specify the portion of incoming to use.

Parameters are:

incoming	A non-null byte array containing the encoded bytes from the client.
offset	The starting position at incoming of the bytes to use.
len	The number of bytes from incoming to use.

4.4.4 wrap

```
public byte[] wrap(byte[] outgoing, int offset, int len)
                   throws SaslException
```

Wraps a byte array to be sent to the client to return the corresponding encoded bytes in a byte array.

This method can be called only after the authentication exchange has completed (i.e., when isComplete() returns true) and only if the authentication exchange has negotiated integrity and/or privacy as the quality of protection; otherwise, a SaslException is thrown. A SaslException is thrown also if outgoing cannot be successfully wrapped.

The result of this method will make up the contents of the SASL buffer as defined in [SASL] without the leading four octet field that represents the length.

offset and len specify the portion of outgoing to use.

Parameters are:

outgoing	A non-null byte array containing the bytes to encode.
offset	The starting position at outgoing of the bytes to use.
len	The number of bytes from outgoing to use.

4.4.5 getMechanismName

```
public String
getMechanismName()
```

Returns the non-null IANA-registered name of the mechanism used by this server, e.g. "GSSAPI" or "CRAM-MD5".

4.4.6 getAuthorizationID

```
public String
```

```
getAuthorizationID() throws SaslException

Reports the authorization ID in effect for the client of this
session. Can only be called if isComplete() returns true; throws
SaslException if called before authentication completes.
```

4.4.7 getNegotiatedProperty

```
public String getNegotiatedProperty(String propName)
                                    throws SaslException

Retrieves the negotiated property. This method can be called only
after the authentication exchange has completed (i.e., when
isComplete() returns true); otherwise, a SaslException is thrown.

For example, this method may be used to obtained the negotiated raw
send buffer size, quality-of-protection, and cipher strength. See
Section 4.1.7 for a list of standard properties.

This method returns null when the specified property was
not negotiated or is not applicable to this mechanism.
```



```
Parameters:

propName     The non-null property name.
```

4.4.8 dispose

```
public abstract void dispose() throws SaslException

Disposes of any system resources or security-sensitive information
the SaslServer might be using. Invoking this method invalidates the
SaslServer instance. This method is idempotent.
```

4.5 public interface SaslServerFactory

An object implementing this interface can provide a SaslServer. The implementation must be thread-safe and handle multiple simultaneous requests. It must also have a public constructor that accepts no argument.

4.5.1 createSaslServer

```
public SaslServer
createSaslServer(String mechanism,
                 String protocol,
                 String serverName,
                 Hashtable props,
                 javax.security.auth.callback.CallbackHandler cbh)
                 throws SaslException
```

Creates a SaslServer using the mechanism supplied. It returns null if no SaslServer can be created using the parameters supplied. Throws SaslException if it cannot create a SaslServer because of an error.

Returns a possibly null SaslServer which supports the specified mechanism. If null, this factory cannot produce a SaslServer for the specified mechanism.

Parameters are:

mechanism	The non-null IANA-registered name of a SASL mechanism (e.g. "GSSAPI", "CRAM-MD5").
protocol	The non-null string name of the protocol for which the authentication is being performed, e.g "pop", "ldap".
serverName	The non-null fully qualified host name of the server.

props	The possibly null set of properties to be used to select the SASL mechanism and to configure the authentication exchange of the selected mechanism. See the Sasl class for a list of standard properties. Other, possibly mechanism-specific, properties can be included. Properties not relevant to the selected mechanism are ignored.
cbh	The possibly null callback handler to be used by the SASL mechanisms to get further information from the application/library to complete the authentication. For example, a SASL mechanism might require the authentication ID, password and realm from the caller. The authentication ID is requested by using a NameCallback. The password is requested by using a PasswordCallback. The realm is requested by using a RealmChoiceCallback

if there is a list of realms to choose from, and by using a RealmCallback if the realm must be entered.

4.5.2 getMechanismNames

```
public String[]
getMechanismNames(Hashtable props)
```

Returns a non-null array of names of mechanisms supported by this factory that match the specified mechanism selection policies.

Parameters are:

props	The possibly null set of properties used to specify the security policy of the SASL mechanisms. For example, if props includes the Sasl.POLICY_NOPLAINTEXT property with the value "true", then the factory must not return any SASL mechanisms that are susceptible to simple plain passive attacks. See the Sasl class for a complete list of policy properties. Non-policy related properties, if present in props, are ignored.

4.6 public class AuthorizeCallback implements javax.security.auth.callback.Callback

This callback is used by SaslServer to determine whether one entity (identified by an authenticated authentication id) can act on behalf of another entity (identified by an authorization id).

4.6.1 Constructors

```
public AuthorizeCallback(String authnID,
                         String authzID)
```

Parameters are :

authnID	The authentication id
authzID	The authorization id

4.6.2 getAuthenticationID

public String getAuthenticationID()

Returns the authentication id to check.

4.6.3 getAuthorizationID

public String getAuthorizationID()

Returns the authorization id to check.

4.6.4 isAuthorized

public boolean isAuthorized()

Returns true if authorization is allowed, false otherwise.

4.6.5 setAuthorized

public void setAuthorized(boolean ok)

Sets whether authorization is allowed or not.

Parameters are:

ok	true if authorization is to be allowed, false otherwise

4.6.6 getAuthorizedID

public String getAuthorizedID()

Returns the id of the authorized user. If null, this means the authorization failed.

4.6.7 setAuthorizedID

public void setAuthorizedID(String id)

Sets the id of the authorized entity. The method is called by the handler only if the id is different from that returned by getAuthorizationID(). For example, the id might need to be

canonicalized for the environment in which it will be used.

Parameters are:

id	The id of the authorized user

4.7 public class RealmCallback extends javax.security.auth.callback.TextInputCallback

This callback is used by SaslClient and SaslServer to retrieve realm information.

4.7.1 Constructors

public RealmCallback (String prompt)

Constructs a RealmCallback with a prompt.

public RealmCallback (String prompt, String defaultRealm)

Constructs a RealmCallback with a prompt and a default realm.

IllegalArgumentException is thrown if prompt is null or the empty string, or if defaultRealm is empty or null.

Parameters are :

prompt	The non-null prompt to use to request the realm information
defaultRealm	The non-null default realm to use

4.8 public class RealmChoiceCallback extends javax.security.auth.callback.ChoiceCallback

This callback is used by SaslClient and SaslServer to obtain one or more realms given a list of realm choices.

4.8.1 Constructors

```
public RealmChoiceCallback (String prompt,
                            String[]choices,
                            int defaultChoice,
                            boolean multipleSelectionsAllowed)
```

Constructs a RealmChoiceCallback with a prompt, a list of choices and a default choice.

IllegalArgumentException is thrown if prompt is null or the empty string, or if defaultRealm is empty or null.

Parameters are :

prompt	The non-null prompt to use to request the realm
choices	The non-null list of realms to choose from
defaultChoice	The choice to use as the default choice when the list of choices is displayed. It is an index into the choices array.
multipleSelectionsAllowed	Specifies whether or not multiple selections can be made from the list of choices.

4.9 public class SaslException extends IOException

Exception thrown on errors and failures that occur when using SASL.

4.9.1 Constructors

```
public SaslException()
```

Constructs a new instance of SaslException. The root exception and the detailed message are null.

```
public SaslException(String message)
```

Constructs a default exception with a detailed message and no root exception.

```
public SaslException(String message,
                     Throwable ex)
```

Constructs a new instance of SaslException with a detailed message and a root exception. For example, a SaslException might result from a problem with the callback handler, which might throw a NoSuchCallbackException if it does not support the requested callback, or throw an IOException if it had problems obtaining data for the callback. The SaslException's root exception would then be the exception thrown by the callback handler.

Parameters are:

message	Possibly null additional detail about the exception.
ex	A possibly null root exception that caused this exception.

4.9.2 getCause

```
public Throwable
getCause()
```

Returns the cause of this exception or null if the cause is nonexistent or unknown. The cause is the throwable that caused this exception to be thrown.

4.9.3 printStackTrace

```
public void
printStackTrace()
```

Prints this exception's stack trace to System.err. If this exception has a root exception, the stack trace of the root exception is also printed to System.err.

```
public void
printStackTrace(PrintStream ps)
```

Prints this exception's stack trace to a print stream. If this exception has a root exception, the stack trace of the root exception is also printed to the print stream.

```
public void
printStackTrace(PrintWriter pw)
```

Prints this exception's stack trace to a print writer. If this exception has a root exception, the stack trace of the root exception is also printed to the print writer.

Parameters are:

ps	The non-null print stream to which to print.
pw	The non-null print writer to which to print.

5. Security Considerations

When SASL authentication is performed over unsecured connections, it is possible for an active attacker to spoof the server's protocol-specific indication that authentication is complete. Clients should protect against this attack by verifying the completion of authentication with the Mechanism Driver by calling the driver's isComplete() method.

Additional security considerations are discussed in [SASL].

6. Copyright

MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

7. Bibliography

[JAAS] Java Software, Sun Microsystems, Inc., "Java Authentication and Authorization Service, "http://java.sun.com/products/jaas", Jan 2000.

[SASL] J. Myers, "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997

8. Authors' Addresses

Rob Weltman
Coscend Corp.
3290 W. Bayshore Road
Palo Alto, CA 94303
+1 650 461 1708
robw@coscend.com

Rosanna Lee
Sun Microsystems
Mail Stop UCUP02-206
901 San Antonio Road
Palo Alto, CA 94303
USA
Email: rosanna.lee@eng.sun.com

9. Acknowledgements

Rob Earhart, then of Carnegie Mellon University, was a coauthor of an earlier draft.

Scott Seligman of Sun Microsystems, Inc. contributed to the architecture and API proposed in this document.

Joe Salowey of WRQ, Anthony J. Nadalin of IBM, Bob Naugle of Bluestone, and Timothy Martin of Carnegie Mellon University contributed to the contents of this revision of the draft through participation in the expert group for Java Specification Request 28 - http://java.sun.com/aboutJava/communityprocess/jsr/jsr_028_sasl.html.

10. Changes from draft-weltman-java-sasl-04.txt

SaslClient, SaslServer

Added dispose() to allow security-sensitive information to be purged from Mechanism Drivers deterministically.

Replaced getInputStream() and getOutputStream() with unwrap() and wrap(), respectively, to remove dependency on stream-based I/O.

Replaced getNegotiatedQop() and getNegotiatedStrength() with the more generic getNegotiatedProperty() to support access to other negotiated properties.

AuthorizeCallback

Changed getAuthenticationId() to getAuthenticationID(), getAuthorizationId() to getAuthorizationID(), and getAuthorizedId() to getAuthorizedID(). This was to conform to the naming convention used in the rest of the API.

11. Changes from draft-weltman-java-sasl-03.txt

Sasl

Added getClientFactories() and getServerFactories().
Updated the list of standard properties.
Added the use of the J2SE version 1.3 service provider guidelines for locating SaslClientFactory and SaslServerFactory implementations.

SaslClient

Added getNegotiatedQop() and getNegotiatedStrength()

SaslClientFactory and SaslServerFactory

getMechanismNames() takes a properties Hashtable as argument.

SaslServer

Added getNegotiatedQop() and getNegotiatedStrength()

AuthorizeCallback

New class to allow SaslServer to determine if an identity may be authorized as another identity.

RealmCallback and RealmChoiceCallback

New classes for obtaining realm information.

SaslException

getRootException changed to getCause, in anticipation of a new standard Java API for nested exceptions. Also, printStackTrace prints both the current and the nested exception.

12. Changes from draft-weltman-java-sasl-02.txt

SecurityLayer

The SecurityLayer interface was removed.

SaslClient

createInitialResponse() was removed. evaluateChallenge() accepts an empty challenge and can return an initial response. hasInitialResponse() was added to determine if the mechanism allows for an initial client response.

SaslClient and SaslServer

getSecurityLayer() was replaced with getInputStream() and getOutputStream().

Package names are |-delimited, not space-delimited, in the pkgs properties.

13. Appendix A - Sample Java LDAP program using SASL

```
   /**************************************************************
     It might look like this in LDAP. The Protocol Driver is
     implemented as part of the authenticate method of
     LDAPConnection. If a security layer is negotiated, the Protocol
     Driver creates new input and output streams that use the
     SaslClient to encode and decode any subsequent messages.
    **************************************************************/

       public void authenticate( String dn,
                                 String[] mechs,
                                 Hashtable props,
                                 CallbackHandler cbh )
                                 throws SaslException {

           // Create SASL client to use for authentication
           SaslClient saslClnt = Sasl.createSaslClient(
                        mechs, dn, "ldap", getHost(), props, cbh);

           if (saslClnt == null) {
```

```
            throw new SaslException("SASL client not available");
        }

        String mechName = saslClnt.getMechanismName();

        // Get initial response, if any
        byte[] response = (saslClnt.hasInitialResponse() ?
                           saslClnt.evaluateChallenge(new byte[0]) :
                           null);


        // Create a bind request message, including the initial
        // response (if any), and send it off
        writeRequest( new LDAPSASLBindRequest( dn, mechName,
                                               response ) );

        // Get the server challenge
        LDAPSASLBindResponse msg =
                             (LDAPSASLBindResponse)readResponse();

        // Authentication done?
        while (!saslClnt.isComplete() &&
               (msg.getStatus() == LDAP_SASL_BIND_IN_PROGRESS ||
                msg.getStatus() == LDAP_SUCCESS)) {

            // No, process challenge to get an appropriate next
            // response
            byte[] challenge = msg.getChallenge();
            response = saslClnt.evaluateChallenge( challenge );

            // May be a success message with no further response
            if ( msg.getStatus() == LDAP_SUCCESS) {
```



```
                if ( response != null ) {
                    // Protocol error; supposed to be done already
                    throw new SaslException("Protocol error in " +
                                            "SASL session");
                }
                break; // done
            }

            // Wrap the response in another bind request and send
            // it off
            writeRequest( new LDAPSASLBindRequest( dn, mechName,
                                                   response ) );
            msg = (LDAPSASLBindResponse)readResponse();
        }
```

```
        // Make sure authentication REALLY is complete
        if ( !saslClnt.isComplete() ) {
            // Authentication session hijacked!
            throw new SaslException( "SASL session hijacked!" );
        }

        // Check if a security layer was negotiated
        String qop = saslClnt.getNegotiatedProperty(Sasl.QOP);
        if ( qop != null && (qop.equalsIgnoreCase("auth-int") ||
                             qop.equalsIgnoreCase("auth-conf")) ) {
            setInputStream(
                new SaslInputStream( saslClnt,
                                     getInputStream() );
            setOutputStream(
                new SaslOutputStream( saslClnt,
                                      getOutputStream() );
        }
    }
```