The Java SSH API User's Guide

by Lee David Painter, lee@3sp.com [mailto:lee@3sp.biz]

by Richard Pernavas, richard@3sp.com [mailto:richard@3sp.biz]

The Java SSH API User's Guide

by Lee David Painter, lee@3sp.com [mailto:lee@3sp.biz]
by Richard Pernavas, richard@3sp.com [mailto:richard@3sp.biz]

Note

This document should be viewed as a work in progress. We have released the user guide in its early stages in the hope that others will help to contribute towards the development of this document in the future.

Table of Contents

Chapter 1. Introduction to the Java SSH API (J2SSH)

SSH, the Secure Shell is a popular network security protocol that defines a specification of how to conduct secure communication over an insecure network. The protocol provides strong guarantees that the parties on both ends of the connection are genuine and that data passing over these connections arrives unmodified and unread by eavesdroppers.

J2SSH provides a platform independent and extensible implementation of the SSH2 version 2 protocol. The components have been designed to provide a convenient means to integrate SSH services into new and existing software applications.

About This Document

This document is intended ...

Conventions used in the document

Throught this guide, different styles for text are used to convey a meaning:

- `C:\Program Files\SSHTools` or `/home/johnd/sshtools`

  A filename (either relative or absolute). The filename may contain a symbolic portion. For example, if `$INSTALL_DIR` is referred, this means the location where J2SSH has been installed.

- `SSHTOOLS_HOME`

  An environment variable. Only applicable if your operating system supports the use of environment variables.

- http://www.sshtools.com

  A hyperlink to a resources, usually on the web.

- ```
  /**
   * An example of a segment of code
   */
        System.out.println("Hello World!");
  ```

  A portion of a source (Java, XML or script) usually used as an example. The portion may be a single line, a segment or a complete listing.

- Please take note of this *emphasis* or you may miss something.

  Used to emphasis a block of text for special attention.

Why Use J2SSH?

If you are planning to develop an SSH implementation or simply wish to take advantage of the protocol in a bespoke application - J2SSH provides you with the freedom to focus your development energies upon the solution in hand. All aspects of the protocol are covered within the programming interface, allowing for customized SSH services to be developed using the lower level protocol components, with higher level components that provide secure shell, secure file transfers and port forwarding. Additionally, the SSH extensibility components allow third party developers to provide additional ciphers, key exchange, public key, compression and message authentication algorithms for J2SSH with relative ease due to the ‘pluggable’ architecture employed.
  ```

Feature list

- SSH client component
- SSH server component
- Transport Protocol
  - Client & server components
  - Abstract server framework for developing additional Transport Protocol services
- Authentication protocol operating in both client/server mode
  - Password authentication
  - Public Key authentication
  - Abstract authentication framework for development of new authentication methods
- Connection protocol operating in both client/server mode
  - Session channel
  - TCP/IP forwarding channel
  - Abstract channel framework for the development of new channels
- Public/private key generation
  - DSA
  - RSA
  - Abstract public key for implementation of other public keys
  - IETF SECSH public key format
  - OpenSSH public key format
  - File format framework for implementing new public/private key file formats
- XML configuration
- Port forwarding configurations listing currently active connections
- Dynamic extension 'plug-in' architecture
- Native support for user authentication and process redirection
- LGPL licensed

Installing the binary distribution

During this section we shall refer to the installation directory of Sshtools as $SSHTOOLS_HOME.

Building the source distribution

Building SSHTools Using ANT

The SSHTools source distribution relies on ANT as its build tool. ANT is available from http://jakarta.apache.org/ant/.

ANT requires a build file called build.xml. The SSHTools build.xml has been developed to be modular and will attempt to compile and build SSHTools subprojects ONLY if the relevant source package is accessible within the source tree. This is achieved through our own ConditionalTask ant task.

To build SSHTools from source, you MUST download the core J2SSH source distribution as this is the only file that contains the ANT build script. Each additional source distribution only contains the necessary source files to build its own distribution alongside J2SSH.

Once you have downloaded the source archives for the SSHTools projects that you require, and have a working copy of ANT available in your path. Extract the J2SSH distribution into your chosen directory (we shall refer to this directory as `$INSTALL_DIR`).

Using a command prompt, change to your `$INSTALL_DIR` and issue the following command:

```
ant
```

This should execute the ANT script build.xml and you should see the log output begin to firstly compile the source, and then building the `$INSTALL_DIR/dist/lib/sshtools-j2ssh-VERSION.jar`.

To build any further SSHTools projects - for example SSHTerm - simply extract the source zip/gz file into your `$INSTALL_DIR`. In the case of SSHTerm, this should create the `$INSTALL_DIR/src/com/sshtools/sshterm` directories, and add additional scripts to `$INSTALL_DIR/bin`.

Once you have extracted the files, execute the ant command once again to execute the script. You should notice that the output log indicates that it has detected the SSHTerm directory and is building the binary distribution files. Upon completion, the script should have built the J2SSH jar as well as the `sshtools-sshterm-VERSION.jar`.

Repeat the above process for any other of the SSHTools projects.

Building SSHTools without ANT

If you decide you do not want to build using the ANT build tool and would rather compile the source in your own development environment, simply add all the dependencies found in `$INSTALL_DIR/lib` to your class-path and remove the `$INSTALL_DIR/src/com/sshtools/ant` directory, as this has a dependency on the ANT tool which is not shipped as part of the distribution.

After adding the dependencies, you may then proceed to compile and build the source.

Chapter 2. Getting started

The first thing you will probably want to do is to connect to an SSH server using J2SSH. This is a fairly straightforward procedure using the SshClient class. This class provides access for connecting, authenticating and starting a session channel, which enables you to execute commands or start the users shell.

```
import com.sshtools.j2ssh.SshClient;
```

First of all prepare your application, in this section we will guide you through the basics so for now just a simple try/catch inside the static main method.

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;

public class SshExample() {

    // A buffered reader so we can request information from the user
    private static BufferedReader reader =
                new BufferedReader(new InputStreamReader(System.in));

    public static void main(String args[]) {

        try {

        // Further code will be added here

        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

The next few sections will guide you through making the initial connection, authenticating the user and executing a command or starting the users shell for a simple console based SSH application.

Making the initial connection

To create an SshClient instance import the class into your implementation class file and use the following code to connect to an SSH server on the standard port 22.

```
        SshClient ssh = new SshClient();

        System.out.print("Host to connect: ");
        String hostname = reader.readLine();
        ssh.connect(hostname);
```

When the client connects to the server, the server supplies its public key for the client to verify. You will see that calling SshClient.connect prompts the user within the console to verify the key:

```
The host firestar is currently unknown to the system
The host key fingerprint is: 1028: 69 54 9c 49 e5 92 59 40 5 66 c5 2e 9d 86 af ed
Do you want to allow this host key? [Yes|No|Always]:
```

In the default implementation J2SSH reads the *$HOME/.ssh/known_hosts* file to determines to which hosts connections may be allowed or denied. This is provided by the class ConsoleKnownHostsKeyVerification and the default behaviour can be emulated by the following code:

```
import com.sshtools.j2ssh.transport.ConsoleKnownHostsKeyVerification;
...
SshClient ssh = new SshClient();
ssh.connect("firestar", new ConsoleKnownHostsKeyVerification());
```

Host key verification is discussed in more detail in Chapter 6. When the connect method returns, the protocol has been negotiated and key exchange has taken place, leaving the connection ready for authenticating the user.

Authenticating the user

Once the connection has been completed the user is required to provide a set of credentials for authentication. All client side authentication methods are implemented using the abstract class:

```
import com.sshtools.j2ssh.authentication.SshAuthenticationClient.
```

To perform authentication, the SshClient class provides the following method:

```
    public int authenticate(SshAuthenticationClient auth);
```

There are currently three authentication methods implemented by J2SSH, 'password', 'publickey' and 'keyboard-interactive'.

Password Authentication

Password authentication is ideal for first time users as it requires no additional configuration within the SSH client or server. The user simply supplies his username and password to the client which is then transmitted over the encrypted connection to the server. The server then checks that the given password is acceptable to the native password-authentication mechanism of the host operating system and returns the result to the client.

J2SSH implements the 'password' authentication method with the following class:

```
import com.sshtools.j2ssh.authentication.PasswordAuthenticationClient
```

Using the password authentication method is straight forward; create an instance of the PasswordAuthentication class, set the username and password and pass to the SshClient to complete the authentication.

```
    /**
     * Create a PasswordAuthentication instance, set the properties
     * and pass to the SessionClient to authenticate
     */
        PasswordAuthenticationClient pwd = new PasswordAuthenticationClient();

        System.out.print("Username: ");
        String username = reader.readLine();
        auth.setUsername(username);

        System.out.print("Password: ");
        String password = reader.readLine();
        auth.setPassword(password);

        int result = ssh.authenticate(pwd);
```

Public-Key Authentication

This method of authentication uses public-key cryptography to verify the user's identity. The user can access an account on an SSH server by proving that they possess a private key. The key is authorized if it is contained within the user's authorization file.

J2SSH implements the 'public-key' authentication method with the following class:

```
import com.sshtools.j2ssh.authentication.PublicKeyAuthenticationClient;
```

Public key authentication requires a little more configuration. First you will require a key pair which can be generated using the ssh-keygen tool, located in the bin directory of the J2SSH installation.

```
C:\sshtools-j2ssh\bin>ssh-keygen -b 1024 -t dsa mykey
```

This command creates a 1024 bit DSA key pair and saves the private key as `mykey` and the public key as `mykey.pub`. Enter the private key passphrase when prompted or simply press return if you don't want the file encrypted with a passphrase. You will be prompted to confirm again that you don't want any passphrase protecting the private key file.

The ssh-keygen tool outputs the public key file in the IETF-SECSH Public Key file format. This is suitable for most SSH servers, however you may need to convert the file if you are using OpenSSH. You can convert the file using the following command:

```
C:\sshtools-j2ssh\bin>ssh-keygen -e mykey.pub > mykey.open
```

This command converts the public key file `mykey.pub` into a new file `mykey.open`. Similary, it is possible to convert an OpenSSH file into an IETF-SECSH file using the -e switch.

To use this key to gain access to your SSH server, you must configure the server to allow access using the key. Refer to your server documentation on how to configure your specific server, however this normally involves the configuration of an authorization file; either stored in the server installation directory or alternatively in your `$USER_HOME/.ssh` directory. The following is a typical example of an authorization file:

```
#SSH Authorization file
key mykey.pub
```

Once your server is configured you can connect using your newly generated key with the following code:

```
import com.sshtools.j2ssh.transport.publickey.SshPrivateKey;
import com.sshtools.j2ssh.transport.publickey.SshPrivateKeyFile;
import com.sshtools.j2ssh.transport.publickey.SshtoolsPrivateKeyFormat;
import com.sshtools.j2ssh.transport.publickey.SshPrivateKey;

..
..

    /**
     * Authenticate using a public key
     */
        PublicKeyAuthenticationClient pk = new PublicKeyAuthenticationClient();

  // Get the username
        System.out.print("Username: ");
        String username = reader.readLine();
        pk.setUsername(username);

  // Open up the private key file
        System.out.print("Path to key: ");
        String keyfile = reader.readLine();
        SshPrivateKeyFile file = SshPrivateKeyFile.parse(
             new File(keyfile));

  // Get the key
       System.out.print("Enter passphrase: ");
       String passphrase = reader.readLine();
       SshPrivateKey key = file.toPrivateKey(passphrase);

  // Set the key and authenticate
        pk.setKey(key);
        int result = ssh.authenticate(pk);
```

For an empty passphrase simply provide a zero length string or **null**. You can also determine whether a private key file is encrypted by calling the following method on the SshPrivateKeyFile instance.

```
        public boolean isPassphraseProtected();
```

Keyboard-interactive Authentication

The 'keyboard-interactive' method is a general purpose authentication mechanism for the SSH protocol, suitable for interactive authentications where the authentication data should be entered via a keyboard. The goal of this

method is to allow an SSH client to support a whole class of authentication mechanisms without knowing the specifics of the actual authentication implemenation. The most common use for this method is to provide password authentication and so this section will assume that the server provides password authentication over keyboard-interactive.

J2SSH implements the keyboard-interactive authentication method with the following class:

```
import com.sshtools.j2ssh.authentication.KBIAuthenticationClient;
```

This method works by providing a callback interface to the authentication subsystem so that the server can request information from the user. Any number of prompts are returned with the name of the authentication mechanism and instructions to display to the user.

```
import com.sshtools.j2ssh.authentication.KBIAuthenticationClient;
import com.sshtools.j2ssh.authentication.KBIPrompt;
import com.sshtools.j2ssh.authentication.KBIRequestHandler;

..
..


 /**
 * Create the keyboard-interactive instance
 */
    KBIAuthenticationClient kbi = new KBIAuthenticationClient();

   // Set the callback interface
    kbi.setKBIRequestHandler(new KBIRequestHandler() {

     public void showPrompts(String name, String instructions, KBIPrompt[] prompts) {
   // Print out the name and instructions
      System.out.println(name);
      System.out.println(instructions);

   // Iterate through the prompts showing one at a time
      String response;
      if(prompts!=null) {
        for(int i=0;i<prompts.length;i++) {
           System.out.print(prompts[i].getPrompt() + ": ");
           try {
             response = reader.readLine();
             prompts[i].setResponse(response);
           }
           catch (IOException ex) {
             ex.printStackTrace();
           }
        }
      }
   // Completed entering the prompts
   }
});

int result = ssh.authenticate(kbi);
```

Hostbased Authentication

The Hostbased authentication method provides a quick but much less secure method of authenticating on the remote server. An SSH server can be configured to allow a client to authenticate based on the host key of the client computer. Whilst this configuraiton varies according to server implementation, J2SSH implements a simple authentication mechanism for hostbased access.

```
import com.sshtools.j2ssh.authentication.HostbasedAuthenticationClient;


HostbasedAuthenticationClient hb = new HostbasedAuthenticationClient();

// Get the username
System.out.print("Username: ");
String username = reader.readLine();
```

```
hb.setUsername(username);

SshPrivateKeyFile file = SshPrivateKeyFile.parse(
             new File("/etc/ssh/server_host_key"));

// Load the host key without a passphrase
hb.setKey(file.toPrivateKey(null));

int result - ss.authenticate(hb);
```

Retrieving the available authentication Methods

It is possible at any time after the connection has been established to request a list of authentication methods that can be used. The getAvailableAuthMethods() function returns a list of authentication method names.

```
        public List getAvailableAuthMethods(String username);
```

It should be noted that the SSH specification allows the server to return authentication methods that are not valid for the user.

The Authentication Result

When the authentication method completes it returns the result of the authentication. This int value can be any of the following three values defined in the class:

```
import com.sshtools.j2ssh.authentication.AuthenticationProtocolState;

..
..
        if(result==AuthenticationProtocolState.FAILED)
            System.out.println("The authentication failed");

        if(result==AuthenticationProtocolState.PARTIAL)
            System.out.println("The authentication succeeded but another"
                               + "authentication is required");

        if(result==AuthenticationProtocolState.COMPLETE)
            System.out.println("The authentication is complete");
```

Prompting the User for Authentication Details?

Each SshAuthenticationClient implementation can optionally be set a prompt interface which allows the user to be prompted for the information once the authenticate method has been invoked.

```
public interface SshAuthenticationPrompt {
  public boolean showPrompt();

  public void setInstance(SshAuthenticationClient instance) throws
      AuthenticationProtocolException;
}
```

The setInstance method is called when the prompt is set and so the instance type should be verified and an exception thrown if the instance cannot be used with the prompt (for example you cannot perform public key authentication with a password prompt!). The instance should then be saved and when the showPrompt method is called, the user is duly prompted for the information and the instance is set with the user's information. Once complete, the prompt returns true to indicate that the user successfully entered correct information.

There are several prompts provided in the J2SSH common packages that provide useful Swing based dialogs to prompt the user.

```
    import com.sshtools.common.authentication.PasswordAuthenticationDialog;

    /**
```

```
     * Create a PasswordAuthenticationDialog instance and call the
     * showAuthenticationMethod so the user can graphically
     * enter their username and password
     */
    PasswordAuthenticationClient pwd = new PasswordAuthenticationClient();

    PasswordAuthenticationDialog dialog = new PasswordAuthenticationDialog(
                                                parent /* insert your parent frame or dialog here
    pwd.setAuthenticationPrompt(dialog);

    int result;

    result = ssh.authenticate(pwd);
```

A Public key authentication prompt is also available:

```
import com.sshtools.common.authentication.PublicKeyAuthenticationPrompt;
```

The Authentication Banner Message

After the initial connection has been made, the server may send an authentication banner message which should be shown to the user prior to authentication. Use the getAuthenticationBanner() method to retrieve the banner message. If no message has been received this method returns an empty string.

```
public String getAuthenticationBanner();
```

Setting Up The Session

Once the user is authenticated you can perform your required tasks, this section describes how to start a session and execute a command on the remote computer. To get a session call the SshClient method:

```
public SessionChannelClient openSessionChannel();
```

This method returns a SessionChannelClient which provides access to the SSH protocol's session channel. Using this instance you can start the users shell, execute commands or start an SSH subsystem.

Setting Environment Variables

Sets the provided environment variable before execution of a command or shell.

```
public boolean setEnvironmentVariable(String name, String value);
```

Requesting a Pseudo Terminal

Requests that the server allocate a pseudo terminal with the given terminal dimensions and terminal type.

```
public boolean requestPseudoTerminal(PseudoTerminal term);
```

Starting the Users Shell

Starts the user's shell.

```
public boolean startShell();
```

Starting an SSH Subsystem

Executes the given SSH subsystem. When starting a subsystem you will be required to read and write the data in the subsystems specified message format. This method does not provide you with an interface into a subsystem client but rather allows you to implement such a client. An example of an SSH subsystem is SFTP, which is discussed later in the document.

```
public boolean startSubsystem(String subsystem);
```

Executing a Command

Executes the given command.

```
public boolean executeCommand(String command);
```

Important

This does not execute a shell command. You cannot for instance issue the command "executeCommand("dir")" on the Windows Operating system as this is a shell command, instead use "cmd.exe /C dir". This method executes a binary executable and so should be used to execute any program other than the users shell.

Handling Session Data

Once the session has been configured and a command, shell or subsystem has been started, you can begin to transfer data to and from the remote computer using the sessions IO streams. These streams provide you with a standardized interface for reading and writing the data.

The Session Channel's OutputStream

The format of writing data varies according to how you configured the session, for example if you executed the users shell then the data should be written as if the user had entered the commands interactively.

```
/** Writing to the session OutputStream */
    OutputStream out = session.getOutputStream();
    String cmd = "ls\n";
    out.write(cmd.getBytes());
```

The Session Channel's InputStream

```
/**
 * Reading from the session InputStream
 */
    InputStream in = session.getInputStream();

    byte buffer[] = new byte[255];
    int read;
    while((read = in.read(buffer)) > 0) {
        String out = new String(buffer, 0, read);
        System.out.println(out);
    }
```

Reading from stderr

The session also provides the stderr data provided by the remote session. Again an InputStream is provided.

```
public InputStream session.getStderrInputStream();
```

Closing the Session

The session can be closed using the following method:

```
public void close();
```

Closing the session does not terminate the connection automatically, although some servers will do so immediately.

Terminating the Connection

The connection can be terminated by either side. To terminate the connection call the SshClient method:

```
public void disconnect();
```

Advanced Client Connectivity

Each SSH connection has a number of options that can be configured to your preference.

Configuring The Connection

Each SSH connection has a number of components, they include encryption ciphers, message authentication algorithms and compression settings. The SSH protocol states that these components must run independently of each other in both directions on the connection. The following class makes it possible to configure these settings:

```
import com.sshtools.j2ssh.configuration.SshConnectionProperties
```

When using the SshClient connect method, it is possible to pass an SshConnectionProperties instance instead of a hostname.

```
SshConnectionProperties properties = new SshConnectionProperties();

properties.setHost("firestar");
properties.setPort(22);

ssh.connect(properties);
```

There are additional methods to set the preffered ciphers:

```
// Sets the prefered client->server encryption cipher
properties.setPrefCSEncryption("blowfish-cbc");

// Sets the preffered server->client encryption cipher
properties.setPrefSCEncryption("3des-cbc");
```

The parameter passed should be the name of the SSH cipher that you require, this can be any installed cipher, the following are currently supported.

- 3des-cbc
- blowfish-cbc
- twofish256-cbc*

- twofish196-cbc*
- twofish128-cbc*
- aes256-cbc*
- aes196-cbc*
- aes128-cbc*
- cast128-cbc*

  * Installed as part of the bouncy castle cipher extensions package

In the same way you can set the message authentication algorithms for each direction of the connection.

```
// Sets the preffered client->server message authenticaiton
properties.setPrefCSMac("hmac-sha1");

// Sets the preffered server->client message authentication
properties.setPrefSCMac("hmac-md5");
```

The following message authentication algorithms are currently supported:.

- hmac-sha1
- hmac-sha1-96
- hmac-md5
- hmac-md5-96

You can set the preffered server host key for server authentication using

```
// Set the preffered server host key
properties.setPrefPublicKey("ssh-rsa");
```

There following public key algorithms are supported

- ssh-dss DSA public keys
- ssh-rsa RSA public keys

Connection Profiles

J2SSH includes a connection profile class that is more useful for more advanced purposes such as graphical frontends and automating authetication procedures.

```
import com.sshtools.j2ssh.util.SshtoolsConnectionProfile;
```

This class can output to XML and so provides a mechanism for storing connection details. It has a number of useful methods that allow application level settings to be stored alongside the existing SSH connection properties.

Monitoring the Connection State

It is possible to monitor the connection state in order to determine whether a session is active, or disconnected. A number of other connection states may also be tested for.

```
import com.sshtools.j2ssh.transport.TransportProtocolState;
import com.sshtools.j2ssh.SshClient;
...
SshClient ssh = new SshClient();
...
TransportProtocolState state = ssh.getConnectionState();
if (state.getValue()==TransportProtocolState.DISCONNECTED) {
  System.out.println("Transport protocol has disconnected!");
}
```

The following states are available, you will be mostly concerned with the connected and disconnected states, but it is possible for the protocol to not have a CONNECTED state but still be connected. When trying to determine the current connection state it should always be evaluated against the DISCONNECTED state.

```
    /**
     *  the transport protocol is uninitialized
     */
    public final static int UNINITIALIZED = 1;

    /**
     *  the transport protocol is connected and negotiating the protocol version
     */
    public final static int NEGOTIATING_PROTOCOL = 2;

    /**
     *  the transport protocol is performing key exchange
     */
    public final static int PERFORMING_KEYEXCHANGE = 3;

    /**
     *  the transport protocol is connected
     */
    public final static int CONNECTED = 4;

    /**
     *  the transport protocol is disconnected
     */
    public final static int DISCONNECTED = 5;
```

It is possible to wait for a specific state, there are a number of methods available that will cause the current thread to wait until the requried state has been reached.

```
// To wait for a specific state
state.waitForState(TransportProtocolState.DISCONNECTED);

// To wait for a state update
state.waitForStateUpdate();
```

Verifying the Server Host Key

When the client connects to the server, the server supplies its public key for the client to verify. You will have already seen that calling SshClient.connect prompts the user within the console to verify the key:

```
The host firestar is currently unknown to the system
The host key fingerprint is: 1028: 69 54 9c 49 e5 92 59 40 5 66 c5 2e 9d 86 af ed
Do you want to allow this host key? [Yes|No|Always]:
```

In the default implementation J2SSH reads the $HOME/.ssh/known_hosts file to determine which hosts connections may be allowed or denied. This is provided by the class ConsoleKnownHostsKeyVerification and the default behaviour can be emulated by the following code:

```
import com.sshtools.j2ssh.transport.ConsoleKnownHostsKeyVerification;
...
SshClient ssh = new SshClient();
ssh.connect("firestar", new ConsoleKnownHostsKeyVerification());
```

The ConsoleKnownHostsKeyVerification class provides a simple console request mechanism to allow for host key verification.

An addtional mechanism is available for swing applicaitons which is implemented by the DialogKnownHostsKeyVerfication class. This prompts the user using a standard JOptionPane dialog.

```
import com.sshtools.j2ssh.transport.DialogKnownHostsKeyVerification;
```

If you prefer to ignore the host key verification process you can use the following class:

```
import com.sshtools.j2ssh.transport.IgnoreHostKeyVerification;
```

To override the default behaviour with your own mechanism, there are two choices. First you can extend the AbstractKnownHostsKeyVerification class. This provides persistence to known_hosts and is recommended for user with additonal implementations when these default styles are not appropriate. For a more simpler method with no persitence, you can implement the base interface for host verification.

```
public interface HostKeyVerification {
  public boolean verifyHost(String host, SshPublicKey pk) throws TransportProtocolException;
}
```

Port Forwarding

Port forwarding allows you to transparently secure another applications data stream by intercepting service requests on one side of the SSH connection, and forwarding them to the recipient at the other side. This is useful in circumstances where you wish to secure the communications of an inherently insecure network application, for example Telnet or SMTP. Once the specifics of the port forward are established through J2SSH, the secured application may commence communication as normal, completely unaware of the underlying forwarding mechanism. Any TCP/IP traffic occuring on the forwarded port is redirected through the SSH session - this is particularly advantageous in circumstances where certain protocols are required to pass through a firewall whose rules restrict their direct usage.

In J2SSH, the following class allows the configuration of port forwarding:

```
import com.sshtools.j2ssh.forwarding.ForwardingClient;
```

Local Forwarding

Local forwarding is one of the two variations of forwarding used by the SSH protocol. By setting up local forwarding, you are specifying that requests initiated from the local machine are to be redirected over the SSH communications channel and delivered to the corresponding port at the other side of the connection. To initiate a local forward in J2SSH you must do the following after obtaining an authenticated SSH session:

```
ForwardingClient forwarding = ssh.getForwardingClient();

// Configure forwarding on local port 10009 to remote port 10007 on mars.sshtools.org
forwarding.addLocalForwarding("Test Local", "0.0.0.0", 10009, "mars.sshtools.org", 10007);

// Starts the specified port forward
forwarding.startLocalForwarding("Test Local");
```

Remote Forwarding

Remote forwarding is similar to local forwarding except that the forwarded connection is initiated from the remote side. This method should be used when the application client requiring secured communications is residing at the remote location (the SSH server side), and the application server is located at the SSH client side. Initiating a remote forward can be done in a similar manner:

```
ForwardingClient forwarding = ssh.getForwardingClient();

// Forward remote port 8081 on mars.sshtools.org to local port 8080
forwarding.addRemoteForwarding("Test Remote", "0.0.0.0", 8081, "mars.sshtools.org", 8080);

forwarding.startRemoteForwarding("Test Remote");
```

Once set up, the example local and remote forwardings may be removed by specifying:

```
forwarding.removeLocalForwarding("Test Local");
forwarding.removeRemoteForwarding("Test Remote");
```

SFTP Client Connectivity

SFTP is an interactive file transfer protocol which performs all operations over the SSH transport, it is a replacement for the original SCP (Secure Copy) protocol that existed in SSH1. It is highly recommended that SFTP be used to perform file transfers in preference to the legacy FTP protocol as authentication details are transmitted in plain-text format with the latter, and as such may be compromised through "password sniffing" attacks.

The following J2SSH namespaces must be imported into a class wishing to incorporate SFTP functionality.

```
import com.sshtools.j2ssh.session.SessionChannelClient;
import com.sshtools.j2ssh.sftp.*;
```

Making the connection

In order to transfer files through SFTP, you must first open an SftpClient connection

```
SessionChannelClient session = ssh.openSessionChannel();
SftpSubsystemClient sftp = new SftpSubsystemClient();
session.startSubsystem(sftp);
```

Reading a file

In order to read a file you should use the openFile method which returns an object of type SftpFile.

```
SftpFile file = sftp.openFile("read.txt", SftpSubsystemClient.OPEN_READ);
```

With the returned file instance you can open an InputStream to read the file

```
// Copy the file to the local computer
SftpFileInputStream in = new SftpFileInputStream(file);
```

```
FileOutputStream out = new FileOutputStream("read.txt");
int read;
byte[] buffer = new byte[4096];
do {
   read = in.read(buffer);
   if(read > 0)
      out.write(buffer,0,read);
} while(read!=-1);

in.close();
out.close();
```

Writing to a file

The following code shows how to create a file on an SFTP server and input some data into it.

```
// Now try to write to a file without creating it!
SftpFile file = sftp.openFile("write.txt",
                SftpSubsystemClient.OPEN_CREATE |
                SftpSubsystemClient.OPEN_WRITE);
// note how the permissions are set after we open
FileAttributes attrs = file.getAttributes();
attrs.setPermissions("rwxrwxrwx");
sftp.setAttributes(file, attrs);

SftpFileOutputStream out = new SftpFileOutputStream(file);
String line = "All work and no play makes johnny a bad boy!\n";

for(int i=0;i<100;i++)
  out.write(line.getBytes());

out.close();
```

I*t is important to remember that when you use the openFile method with the OPEN_CREATE flag, permissions will not be as expected if you pass a file attributes object when creating. Currently the best method is call getAttributes to obtain the attributes straight after the file creation, change the permissions on the attribute object and finally, call setAttributes to apply these.*

Changing file permissions

In order to change permissions associated with files, we need to use the FileAttributes class to specify permissions in the standard Unix format. Place code similar to the following after the openFile method returns.

```
// Setting attributes on an open file
... //open the file
sftp.changePermissions(file, "rwxr--r--");

// Changing permissions on an unopened file
sftp.changePermissions("write.txt", "rwxr--r--");
```

Directory operations

The SftpSubsystemClient API provides a range of directory manipulation methods. The following code shows examples of some of these. Directories are treated in a similar manner to files, using the SftpFile object for both. You may use the isDirectory method of this object to determine whether the object is a directory or a file.

```
// Get the users home directory as specified in the SFTP server configuration
String defaultDir = sftp.getDefaultDirectory();

// Open the home directory
SftpFile dir = sftp.openDirectory(defaultDir);

// Read the directories children
Vector children = new Vector();
```

```
while(sftp.listChildren(dir, children) > 0);

// Make a new directory in the home dir
sftp.makeDirectory("foo");

// Remove the directory
sftp.removeDirectory("foo");
```

Chapter 3. SSH Daemon

Coming soon

Implementing Native Platform Features

Coming soon

Chapter 4. J2SSH Extensions

The architecture utilized in J2SSH allows for the ease of development of derivative software. This feature is of particular importance to developers since it simplifies the development and integration of third party add-ons using the library. Any application using the API instantly benefits from the ability to use all the other available API extensions by default.

Installing Extensions

You can install a set of extensions by placing the jar file in the `$SSHTOOLS_HOME/lib/ext` directory. All jars in this directory are automatically added to the J2SSH classpath. Further configuration is then needed to determine which extensions you would like to activate, this is typically acheived through adding additional XML elements in the sshtools.xml configuration file.

Ciphers

Cipher extensions allow for the expansion of the cryptographical algorithms used by J2SSH. A number of additional ciphers are available in addition to the standard set, available from the www.sshtools.com website, compliments of the Legion of the Bouncy Castle.

The additional ciphers are:

- cast128-cbc
- twofish256-cbc
- twofish196-cbc
- twofish128-cbc
- aes256-cbc
- aes196-cbc
- aes128-cbc

Chapter 5. Key Files

A key file is also known as an SSH identity. It is a mechanism allowing for verification of the identity of an SSH client or server. An SSH identity takes the form of a cryptographic key pair - a public key and a private key.

Since there is currently no formal specification for the SSH *private keys*, a number of different key file formats exist for the process of user authentication. SSHTools currently uses its own encrypted key format, however a conversion tool exists for converting files from the IETF *public key* format to the OpenSSH public key format so that these can be included within OpenSSH authorization files. The formatting tools now allow for the use of additional key formats which can be added to the library via the sshtools.xml file as long as a simple formatting class is implemented. This will allow SSHTools to read other private key files as soon as formatting implementations become available.

The key generation utility allows four operations to be performed:

- Generate a new key pair using either RSA or DSA encyption algorithms
- Convert IEFT SECSH public key file to OpenSSH file format
- Convert OpenSSH public key file to IEFT SECSH format
- Change passphrase

Public Key Files

IETF SECSH

The IETF key format is the public key file format that SSHTools, along with many of the proprietary SSH vendors adhere to. As such these keys may be used within SSHTools without further modification.

OpenSSH

OpenSSH public key files may be converted for use within J2SSH by using the keygen utility.

Please note, the conversion of OpenSSH private key files is not currently supported.

Implementing New Public Key Formats

How to implement a new key format

Private Key Files

Explain the private key format

Sshtools Private Key Format

The sshtools private key format

Implementing New Private Key Formats

How to implement a private key format

Chapter 6. Configuration

J2SSH is very flexible and is configured through a series of XML files. This section looks at each file to explain the configuration settings. The Configuration settings are managed through the ConfigurationLoader class and in most instances the default behaviour of this class will be acceptable to most implementations.

```
import com.sshtools.j2ssh.configuration.ConfigurationLoader;
```

When running Sshtools within an application, the ConfigurationLoader requires the sshtools.home system property to be set, this can be achieved 2 different ways.

1. In your VM parameters use:

   ```
   -Dsshtools.home=$SSHTOOLS_HOME
   ```

2. From within your application use:

   ```
   System.setProperty(“sshtools.home”, “$SSHTOOLS_HOME”);
   ```

However, when using Sshtools within an applet it may not be possible to set system properties so an alternative method has been provided which can be used from either an applet or an application implementation. This involves

API Configuration

Sshtools employs a plug-in architecture that allows ease of integration of new Sshtools components. The API configuration file is used to add or override the default implementations of Ciphers, Message Authentications, Authentication Methods, Public Key Mechanisms and Key Exchange Methods and to specify extension libaries that are dynamically loaded at runtime.

The default configuration file sshtools.xml can be found in the `$SSHTOOLS_HOME/conf` directory.

Allowed/Denied Hosts

When each SSH connection is established the Transport Protocol performs an authentication of the server. The server passes it’s public key to the client with a signature of data. The client verifies that the signature is valid and that the host is allowed to connect with the client.

J2SSH verifies the host key by calling an instance of the abstract class HostKeyVerification as described in the earlier section Advanced Connectivity. This class manages a host verification file which defaults to hosts.xml. Although the various implementations of this abstract class manage the host file, it is possible to manually configure.

```
<?xml version="1.0" encoding="UTF-8" ?>

<HostAuthorizations>
    <AllowHost HostName="Firestar" Fingerprint="1024: 69 54 9c.../>
    <DenyHost Hostname="Phobos"/>
</HostAuthorizations>
```

Server Configuration

Explain about server configuration

Platform Configuration

Explain about platform configuration

Appendix A. GNU Free Documentation License

Version 1.2, November 2002

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or ab-

sence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any

large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copy-

right holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warrany Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.