## Port forwarding

### PortForwardingEventListener

Informs and allows tracking of port forwarding events as described in [RFC 4254 - section 7](https://tools.ietf.org/html/rfc4254#section-7) as well as the (simple) [SOCKS](https://en.wikipedia.org/wiki/SOCKS) protocol (versions 4, 5). In this context, one can create a PortForwardingTracker that can be used in a try-with-resource block so that the set up forwarding is automatically torn down when the tracker is close()-d:

client.addPortForwardingEventListener(new MySuperDuperListener());  
  
 try (ClientSession session = client.connect(user, host, port).verify(...timeout...).getSession()) {  
 session.addPasswordIdentity(password);  
 session.auth().verify(...timeout...);  
  
 try (PortForwardingTracker tracker = session.createLocal/RemotePortForwardingTracker(...)) {  
 ...do something that requires the tunnel...  
 }  
  
 // Tunnel is torn down when code reaches this point  
 }

### Standard port forwarding

Port forwarding as specified in [RFC 4254 - section 7](https://tools.ietf.org/html/rfc4254#section-7) is fully supported by the client and server. From the client side, this capability is exposed via the start/stopLocal/RemotePortForwarding method. The key player in this capability is the configured ForwardingFilter that controls this feature - on **both** sides - client and server. By default, this capability is **disabled** - i.e., the user must provide an implementation and call the appropriate setForwardingFilter method on the client/server.

SshClient client = ...create/obtain an instance...  
 client.setForwardingFilter(...filter instance...);  
  
 SshServer server = ...create/obtain an instance...  
 server.setForwardingFilter(...filter instance...);

The code contains 2 simple implementations - an AcceptAllForwardingFilter and a RejectAllForwardingFilter one that can be used for these trivial policies. **Note:** setting a *null* filter is equivalent to rejecting all such attempts.

In order to help with the forwarding policy, the filter is actually made up of 3 "groups" of forwarding:

* AgentForwardingFilter
* X11ForwardingFilter
* TcpForwardingFilter

It is possible to implement each and every one separately and then combine them via ForwardingFilter#asForwardingFilter. In this context, one does not have to implement all 3 - any implementation not provided is assumed to be disabled. Furthermore, there are reasonable default implementations for all 3, so one can override only a specific group policy and provide defaults for the rest.

### SOCKS

The code implements a [SOCKS](https://en.wikipedia.org/wiki/SOCKS) proxy for versions 4 and 5. The proxy capability is invoked via the start/stopDynamicPortForwarding methods.

### Proxy agent

The code provides to some extent an SSH proxy agent via the available SshAgentFactory implementations. As of latest version both [Secure Shell Authentication Agent Protocol Draft 02](https://tools.ietf.org/html/draft-ietf-secsh-agent-02) and its [OpenSSH](https://www.libssh.org/features/) equivalent are supported. **Note:** in order to support this feature the [Apache Portable Runtime Library](https://apr.apache.org/) needs to be added to the Maven dependencies:

<dependency>  
 <groupId>tomcat</groupId>  
 <artifactId>tomcat-apr</artifactId>  
 </dependency>

**Note:** Since the portable runtime library uses **native** code, one needs to also make sure that the appropriate *.dll/.so* library is available in the LD\_LIBRARY\_PATH.