# Extension modules

There are several extension modules available - specifically, the *sshd-contrib* module contains some of them. **Note:** the module contains experimental code that may find its way some time in the future to a standard artifact. It is also subject to changes and/or deletion without any prior announcement. Therefore, any code that relies on it should also store a copy of the sources in case the classes it used it are modified or deleted.

## LDAP adaptors

The *sshd-ldap* artifact contains an [LdapPasswordAuthenticator](https://issues.apache.org/jira/browse/SSHD-607) and an [LdapPublicKeyAuthenticator](https://issues.apache.org/jira/browse/SSHD-608) that have been written along the same lines as the [openssh-ldap-publickey](https://github.com/AndriiGrytsenko/openssh-ldap-publickey) project. The authenticators can be easily configured to match most LDAP schemes, or alternatively serve as base classes for code that extends them and adds proprietary logic.

## Useful extra components in *sshd-contrib*

* InteractivePasswordIdentityProvider - helps implement a PasswordIdentityProvider by delegating calls to UserInteraction#getUpdatedPassword. The way to use it would be as follows:

try (ClientSession session = client.connect(login, host, port).await().getSession()) {  
 session.setUserInteraction(...); // this can also be set at the client level  
 PasswordIdentityProvider passwordIdentityProvider =  
 InteractivePasswordIdentityProvider.providerOf(session, "My prompt");  
 session.setPasswordIdentityProvider(passwordIdentityProvider);  
 session.auth.verify(...timeout...);  
 ... continue with the authenticated session ...  
}

or

UserInteraction ui = ....;  
try (ClientSession session = client.connect(login, host, port).await().getSession()) {  
 PasswordIdentityProvider passwordIdentityProvider =  
 InteractivePasswordIdentityProvider.providerOf(session, ui, "My prompt");  
 session.setPasswordIdentityProvider(passwordIdentityProvider);  
 session.auth.verify(...timeout...);  
 ... continue with the authenticated session ...  
}

**Note:** UserInteraction#isInteractionAllowed is consulted prior to invoking getUpdatedPassword - if it returns *false* then password retrieval method is not invoked, and it is assumed that no more passwords are available

* SimpleAccessControlScpEventListener - Provides a simple access control by making a distinction between methods that upload data and ones that download it via SCP. In order to use it, simply extend it and override its isFileUpload/DownloadAllowed methods
* SimpleAccessControlSftpEventListener - Provides a simple access control by making a distinction between methods that provide SFTP file information - including reading data - and those that modify it
* ProxyProtocolAcceptor - A working prototype to support the PROXY protocol as described in [HAProxy Documentation](http://www.haproxy.org/download/1.8/doc/proxy-protocol.txt)
* ThrottlingPacketWriter - An example of a way to overcome big window sizes when sending data - as described in [SSHD-754](https://issues.apache.org/jira/browse/SSHD-754) and [SSHD-768](https://issues.apache.org/jira/browse/SSHD-768)
* AndroidOpenSSLSecurityProviderRegistrar - A security registrar that uses the [AndroidOpenSSL](https://github.com/guardianproject/openssl-android) security provider
* LegacyDSASigner - A java.security.Signature that applies SHA-1 with DSA keys regardless of their key length - i.e., despite FIPS186-3 section 4.2 that mandates usage of SHA-2 for keys greater than 1024 bits. This is in accordance with RFC 4253 that was never amended to specify any other digest for such keys. The signer can be use to provide a custom implementation of SignatureDSA (and its factory) that uses this signer instead of the JCE or *Bouncycastle* one - see comments on issue [SSHD-945](https://issues.apache.org/jira/browse/SSHD-945).