# SFTP

Both client-side and server-side SFTP are supported. Starting from version 2.0, the SFTP related code is located in the sshd-sftp artifact, so one needs to add this additional dependency to one's maven project:

<dependency>  
 <groupId>org.apache.sshd</groupId>  
 <artifactId>sshd-sftp</artifactId>  
 <version>...same as sshd-core...</version>  
 </dependency>

## Server-side SFTP

On the server side, the following code needs to be added:

SftpSubsystemFactory factory = new SftpSubsystemFactory.Builder()  
 ...with...  
 ...with...  
 .build();  
 server.setSubsystemFactories(Collections.singletonList(factory));

**Note:** the factory uses an ad-hoc CloseableExecutorService in order to spawn the necessary threads for processing the protocol messages. The user can provide a custom Supplier of such a service - however, it must be protected from shutdown if the user needs it to remain active between successive SFTP session. This can be done via the ThreadUtils#noClose utility:

CloseableExecutorService mySpecialExecutor = ...;  
 SftpSubsystemFactory factory = new SftpSubsystemFactory.Builder()  
 .withExecutorServiceProvider(() -> ThreadUtils.noClose(mySpecialExecutor))  
 .build();  
 server.setSubsystemFactories(Collections.singletonList(factory));

### SftpEventListener

Provides information about major SFTP protocol events. The provided File/DirectoryHandle to the various callbacks can also be used to store user-defined attributes via its AttributeStore implementation. The listener is registered at the SftpSubsystemFactory:

public class MySfpEventListener implements SftpEventListener {  
 private static final AttributeKey<SomeType> MY\_SPECIAL\_KEY = new Attribute<SomeType>();  
  
 ...  
 @Override  
 public void opening(ServerSession session, String remoteHandle, Handle localHandle) throws IOException {  
 localHandle.setAttribute(MY\_SPECIAL\_KEY, instanceOfSomeType);  
 }  
  
 @Override  
 public void writing(  
 ServerSession session, String remoteHandle, FileHandle localHandle,  
 long offset, byte[] data, int dataOffset, int dataLen)  
 throws IOException {  
 SomeType myData = localHandle.getAttribute(MY\_SPECIAL\_KEY);  
 ...do something based on my data...  
 }  
 }  
  
  
 SftpSubsystemFactory factory = new SftpSubsystemFactory();  
 factory.addSftpEventListener(new MySftpEventListener());  
 sshd.setSubsystemFactories(Collections.<NamedFactory<Command>>singletonList(factory));

**Note:** the attached attributes are automatically removed once handle has been closed - regardless of whether the close attempt was successful or not. In other words, after SftpEventListener#closed has been called, all attributes associated with the handle are cleared.

### SftpFileSystemAccessor

This is the abstraction providing the SFTP server subsystem access to files and directories. The SFTP subsystem uses this abstraction to obtain file channels and/or directory streams. One can override the default implementation and thus be able to track and/or intervene in all opened files and folders throughout the SFTP server subsystem code. The accessor is registered/overwritten in via the SftpSubSystemFactory:

SftpSubsystemFactory factory = new SftpSubsystemFactory.Builder()  
 .withFileSystemAccessor(new MySftpFileSystemAccessor())  
 .build();  
 server.setSubsystemFactories(Collections.singletonList(factory));

**Note:**

* Closing of file channel/directory streams created by the accessor are also closed via callbacks to the same accessor
* When closing a file channel that may have been potentially modified, the default implementation forces a synchronization of the data with the file-system. This behavior can be modified by setting the sftp-auto-fsync-on-close property to *false* (or by providing a customized implementation that involves other considerations as well).

### Internal exceptions and error message handling

If an exception is thrown during processing of an SFTP command, then the exception is translated into a SSH\_FXP\_STATUS message using a registered SftpErrorStatusDataHandler. The default implementation provides a short description of the failure based on the thrown exception type. However, users may override it when creating the SftpSubsystemFactory and provide their own codes and/or messages - e.g., for debugging one can register a DetailedSftpErrorStatusDataHandler (see sshd-contrib) that "leaks" more information in the generated message.

If the registered handler implements ChannelSessionAware then it will also be informed of the registered ChannelSession when it is provided to the SftpSubsystem itself. This can be used to register an extended data writer that can handle data sent via the STDERR channel. **Note:** this feature is allowed according to [SFTP version 4 - section 3.1](https://tools.ietf.org/html/draft-ietf-secsh-filexfer-04#section-3.1):

Packets are sent and received on stdout and stdin. Data sent on stderr by the server SHOULD be considered debug or supplemental error information, and MAY be displayed to the user.

however, the current code provides no built-in support for this feature.

If registering an extended data writer then one should take care of any race conditions that may occur where (extended) data may arrive before the handler is informed of the existence of the ChannelSession. For this purpose one should configure a reasonable buffer size by setting the channel-session-max-extdata-bufsize property. This way, if any data arrives before the extended data handler is registered it will be buffered (up to the specified max. size). **Note:** if a buffer size is configured but no extended data handler is registered when channel is spawning the command then an exception will occur.

### Symbolic links handling

Whenever the server needs to execute a command that may behave differently if applied to a symbolic link instead of its target it consults the AbstractSftpSubsystemHelper#resolvePathResolutionFollowLinks method. By default, this method simply consults the value of the sftp-auto-follow-links configuration property (default=*true*).

**Note:** the property is consulted only for cases where there is no clear indication in the standard how to behave for the specific command. E.g., the lsetstat@openssh.com specifically specifies that symbolic links should not be followed, so the implementation does not consult the aforementioned property.

## Client-side SFTP

In order to obtain an SftpClient instance one needs to use an SftpClientFactory:

try (ClientSession session = ...obtain session...) {  
 SftpClientFactory factory = ...obtain factory...  
 try (SftpClient client = factory.createSftpClient(session)) {  
 ... use the SFTP client...  
 }  
   
 // NOTE: session is still alive here...  
 }

A default client factory implementations is provided in the module - see SftpClientFactory.instance()

If the intended use of the client instance is "one-shot" - i.e., the client session should be closed when the SFTP client instance is closed, then it is possible to obtain a special wrapper that implements this functionality:

// The underlying session will also be closed when the client is  
try (SftpClient client = createSftpClient(....)) {  
 ... use the SFTP client...  
}  
  
SftpClient createSftpClient(...) {  
 ClientSession session = ...obtain session...  
 SftpClientFactory factory = ...obtain factory...  
 SftpClient client = factory.createSftpClient(session);  
 return client.singleSessionInstance();  
}

### Using a custom SftpClientFactory

The code creates SftpClient-s and SftpFileSystem-s using a default built-in SftpClientFactory instance (see DefaultSftpClientFactory). Users may choose to use a custom factory in order to provide their own implementations - e.g., in order to override some default behavior - e.g.:

SshClient client = ... setup client...  
  
 try (ClientSession session = client.connect(user, host, port).verify(timeout).getSession()) {  
 session.addPasswordIdentity(password);  
 session.auth.verify(timeout);  
  
 // User-specific factory  
 try (SftpClient sftp = MySpecialSessionSftpClientFactory.INSTANCE.createSftpClient(session)) {  
 ... instance created through SpecialSessionSftpClientFactory ...  
 }  
 }

### Version selection via SftpVersionSelector

The SFTP subsystem code supports versions 3-6 (inclusive), and by default attempts to negotiate the highest possible one - on both client and server code. The user can intervene and force a specific version or a narrower range.

SftpVersionSelector myVersionSelector = new SftpVersionSelector() {  
 @Override  
 public int selectVersion(ClientSession session, boolean initial, int current, List<Integer> available) {  
 int selectedVersion = ...run some logic to decide...;  
 return selectedVersion;  
 }  
 };  
  
 try (ClientSession session = client.connect(user, host, port).verify(timeout).getSession()) {  
 session.addPasswordIdentity(password);  
 session.auth.verify(timeout);  
  
 SftpClientFactory factory = SftpClientFactory.instance();  
 try (SftpClient sftp = factory.createSftpClient(session, myVersionSelector)) {  
 ... do SFTP related stuff...  
 }  
 }

**Note:** the version selector is invoked **twice** - the first time in order to retrieve the initial version to be used when estabilishing the SFTP channel, and the second after having done so after receiving the server's version. The invocations are distinguished by the initial parameter value.

On the server side, version selection restriction is more complex - please remember that according to the protocol specification

The server responds with a SSH\_FXP\_VERSION packet, supplying the lowest (!) of its own and the client's version number

Currently at the server we support requiring a **specific** version via the SftpSubsystem#SFTP\_VERSION configuration key. The same can be achieved for the CLI SSHD code by specifying -o sftp-version=N option.

For more advanced restrictions one needs to sub-class SftpSubSystem and provide a non-default SftpSubsystemFactory that uses the sub-classed code.

### Using SftpFileSystemProvider to create an SftpFileSystem

The code automatically registers the SftpFileSystemProvider as the handler for sftp:// URL(s). Such URLs are interpreted as remote file locations and automatically exposed to the user as [Path](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Path.html) objects. In effect, this allows the code to "mount" a remote directory via SFTP and treat it as if it were local using standard [java.nio](https://docs.oracle.com/javase/8/docs/api/java/nio/package-frame.html) calls like any "ordinary" file system.

// Direct URI  
 Path remotePath = Paths.get(new URI("sftp://user:password@host/some/remote/path"));  
 // Releasing the file-system once no longer necessary  
 try (FileSystem fs = remotePath.getFileSystem()) {  
 ... work with the remote path...  
 }  
  
 // "Mounting" a file system  
 URI uri = SftpFileSystemProvider.createFileSystemURI(host, port, username, password);  
 try (FileSystem fs = FileSystems.newFileSystem(uri, Collections.<String, Object>emptyMap())) {  
 Path remotePath = fs.getPath("/some/remote/path");  
 ...  
 }  
  
 // Full programmatic control  
 SshClient client = ...setup and start the SshClient instance...  
 SftpFileSystemProvider provider = new SftpFileSystemProvider(client);  
 URI uri = SftpFileSystemProvider.createFileSystemURI(host, port, username, password);  
 try (FileSystem fs = provider.newFileSystem(uri, Collections.<String, Object>emptyMap())) {  
 Path remotePath = fs.getPath("/some/remote/path");  
 }

The obtained Path instance can be used in exactly the same way as any other "regular" one:

```java

try (InputStream input = Files.newInputStream(remotePath)) {  
 ...read from remote file...  
}  
  
try (DirectoryStream<Path> ds = Files.newDirectoryStream(remoteDir)) {  
 for (Path remoteFile : ds) {  
 if (Files.isRegularFile(remoteFile)) {  
 System.out.println("Delete " + remoteFile + " size=" + Files.size(remoteFile));  
 Files.delete(remoteFile);  
 } else if (Files.isDirectory(remoteFile)) {  
 System.out.println(remoteFile + " - directory");  
 }  
 }  
}

```

It is highly recommended to close() the mounted file system once no longer necessary in order to release the associated SFTP session sooner rather than later - e.g., via a try-with-resource code block.

**Caveat:** Due to URI encoding of the username/password as a basic authentication, the system currently does not allow colon (:) in either one in order to avoid parsing confusion. See [RFC 3986 - section 3.2.1](https://tools.ietf.org/html/rfc3986#section-3.2.1):

Use of the format "user:password" in the userinfo field is deprecated ... Applications may choose to ignore or reject such data when it is received as part of a reference...

#### Configuring the SftpFileSystemProvider

When "mounting" a new file system one can provide extra configuration parameters using either the environment map in the [FileSystems#newFileSystem](https://docs.oracle.com/javase/8/docs/api/java/nio/file/FileSystems.html#newFileSystem) method or via the URI query parameters. See the SftpFileSystemProvider for the available configuration keys and values.

// Using explicit parameters  
 Map<String, Object> params = new HashMap<>();  
 params.put("param1", value1);  
 params.put("param2", value2);  
 ...etc...  
  
 URI uri = SftpFileSystemProvider.createFileSystemURI(host, port, username, password);  
 try (FileSystem fs = FileSystems.newFileSystem(uri, params)) {  
 Path remotePath = fs.getPath("/some/remote/path");  
 ... work with the remote path...  
 }  
  
 // Using URI parameters  
 Path remotePath = Paths.get(new URI("sftp://user:password@host/some/remote/path?param1=value1&param2=value2..."));  
 // Releasing the file-system once no longer necessary  
 try (FileSystem fs = remotePath.getFileSystem()) {  
 ... work with the remote path...  
 }

**Note**: if **both** options are used then the URI parameters **override** the environment ones

Map<String, Object> params = new HashMap<>();  
 params.put("param1", value1);  
 params.put("param2", value2);  
  
 // The value of 'param1' is overridden in the URI  
 try (FileSystem fs = FileSystems.newFileSystem(  
 new URI("sftp://user:password@host/some/remote/path?param1=otherValue1", params)) {  
 Path remotePath = fs.getPath("/some/remote/path");  
 ... work with the remote path...  
 }

### Configuring the client session used to create an SftpFileSystem

It is possible to register a SftpFileSystemClientSessionInitializer with the provider instead of the default one and thus better control the ClientSession used to generate the file-system instance. The default implementation simply connects and authenticates before creating a default SftpFileSystem instance. Users may wish to override some options or provide their own - e.g., execute a password-less authentication instead of the (default) password-based one:

SftpFileSystemProvider provider = ... obtain/create a provider ...  
 provider.setSftpFileSystemClientSessionInitializer(new SftpFileSystemClientSessionInitializer() {  
 @Override  
 public void authenticateClientSession(  
 SftpFileSystemProvider provider, SftpFileSystemInitializationContext context, ClientSession session)  
 throws IOException {  
 /\*  
 \* Set up password-less login instead of password-based using the specified key  
 \*  
 \* Note: if SSH client and/or session already have a KeyPairProvider set up and the code  
 \* knows that these keys are already registered with the remote server, then no need to  
 \* add the public key identitiy - can simply call sesssion.auth().verify(context.getMaxAuthTime()).  
 \*/  
 KeyPair kp = ... obtain a registered key-pair...  
 session.addPublicKeyIdentity(kp);  
 sesssion.auth().verify(context.getMaxAuthTime());  
 }  
 });

### SFTP sent/received names encoding

By default, the SFTP client uses UTF-8 to encode/decode any referenced file/folder name. However, some servers do not properly encode such names, and thus the "visible" names by the client become corrupted, or even worse - cause an exception upon decoding attempt. The SftpClient exposes a get/setNameDecodingCharset method which enables the user to modify the charset - even while the SFTP session is in progress - e.g.:

try (SftpClient client = ...obtain an instance...) {  
 client.setNameDecodingCharset(Charset.forName("ISO-8859-8"));  
 for (DirEntry entry : client.readDir(...some path...)) {  
 ...handle entry assuming ISO-8859-8 encoded names...  
 }  
  
 client.setNameDecodingCharset(Charset.forName("ISO-8859-4"));  
 for (DirEntry entry : client.readDir(...some other path...)) {  
 ...handle entry assuming ISO-8859-4 encoded names...  
 }  
 }

The initial charset can be pre-configured on the client/session by using the sftp-name-decoding-charset property - if none specified then UTF-8 is used. **Note:** the value can be a charset name or a java.nio.charset.Charset instance - e.g.:

SshClient client = ... setup/obtain an instance...  
 // default for ALL SFTP clients obtained through this client  
 PropertyResolverUtils.updateProperty(client, SftpClient.NAME\_DECODING\_CHARSET, "ISO-8859-8");  
  
 try (ClientSession session = client.connect(...)) {  
 session.addPasswordIdentity(password);  
 session.auth().verify(timeout);  
  
 // default for ALL SFTP clients obtained through the session - overrides client setting  
 PropertyResolverUtils.updateProperty(session, SftpClient.NAME\_DECODING\_CHARSET, "ISO-8859-4");  
  
 SftpClientFactory factory = SftpClientFactory.instance();  
 try (SftpClient sftp = factory.createSftpClient(session)) {  
 for (DirEntry entry : sftp.readDir(...some path...)) {  
 ...handle entry assuming ISO-8859-4 (inherited from the session) encoded names...  
 }  
  
 // override the inherited default from the session  
 sftp.setNameDecodingCharset(Charset.forName("ISO-8859-1"));  
  
 for (DirEntry entry : sftp.readDir(...some other path...)) {  
 ...handle entry assuming ISO-8859-1 encoded names...  
 }  
 }  
 }

### SFTP aware directory scanners

The framework provides special SFTP aware directory scanners that look for files/folders matching specific patterns. The scanners support *recursive* scanning of the directories based on the selected patterns.

E.g. - let's assume the layout present below

root  
 + --- a1.txt  
 + --- a2.csv  
 + sub1  
 +--- b1.txt  
 +--- b2.csv  
 + sub2  
 + --- c1.txt  
 + --- c2.csv

Then scan results from root are expected as follows for the given patterns

* "\*\*/\*" - all the files/folders - [a1.txt, a1.csv, sub1, sub2, b1.txt, b1.csv, c1.txt, c2.csv]
* "\*\*/\*.txt" - only the ".txt" files - [a1.txt, b1.txt, c1.txt]
* "\*" - only the files/folders at the root - [a1.txt, a1.csv, sub1, sub2]
* "\*.csv" - only a1.csv at the root

**Note:** the scanner supports various patterns - including *regex* - see DirectoryScanner and SelectorUtils classes for supported patterns and matching - include case sensitive vs. insensitive match.

// Using an SftpPathDirectoryScanner  
 FileSystem fs = ... obtain an SFTP file system instance ...  
 Path rootDir = fs.getPath(...remote path...);  
 DirectoryScanner ds = new SftpPathDirectoryScanner(basedir, ...pattern...);  
 Collection<Path> matches = ds.scan();  
   
 // Using an SftpClientDirectoryScanner  
 SftpClient client = ... obtain a client instance ...  
 SftpClientDirectoryScanner ds = new SftpClientDirectoryScanner(basedir, ...pattern...);  
 Collection<ScanDirEntry> matches = ds.scan(client);

## Extensions

Both client and server support several of the SFTP extensions specified in various drafts:

* supported - [DRAFT 05 - section 4.4](http://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/draft-ietf-secsh-filexfer-05.tx)
* supported2 - [DRAFT 13 section 5.4](https://tools.ietf.org/html/draft-ietf-secsh-filexfer-13#page-10)
* versions - [DRAFT 09 Section 4.6](http://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/draft-ietf-secsh-filexfer-09.txt)
* vendor-id - [DRAFT 09 - section 4.4](http://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/draft-ietf-secsh-filexfer-09.txt)
* acl-supported - [DRAFT 11 - section 5.4](https://tools.ietf.org/html/draft-ietf-secsh-filexfer-11)
* newline - [DRAFT 09 Section 4.3](http://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/draft-ietf-secsh-filexfer-09.txt)
* md5-hash, md5-hash-handle - [DRAFT 09 - section 9.1.1](http://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/draft-ietf-secsh-filexfer-09.txt)
* check-file-handle, check-file-name - [DRAFT 09 - section 9.1.2](http://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/draft-ietf-secsh-filexfer-09.txt)
* copy-file, copy-data - [DRAFT 00 - sections 6, 7](http://tools.ietf.org/id/draft-ietf-secsh-filexfer-extensions-00.txt)
* space-available - [DRAFT 09 - section 9.3](http://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/draft-ietf-secsh-filexfer-09.txt)

Furthermore several [OpenSSH SFTP extensions](https://github.com/openssh/openssh-portable/blob/master/PROTOCOL) are also supported:

* fsync@openssh.com
* fstatvfs@openssh.com
* hardlink@openssh.com
* posix-rename@openssh.com
* statvfs@openssh.com
* lsetstat@openssh.com

On the server side, the reported standard extensions are configured via the SftpModuleProperties.CLIENT\_EXTENSIONS configuration key, and the *OpenSSH* ones via the SftpModuleProperties.OPENSSH\_EXTENSIONS.

On the client side, all the supported extensions are classes that implement SftpClientExtension. These classes can be used to query the client whether the remote server supports the specific extension and then obtain a parser for its contents. Users can easily add support for more extensions in a similar manner as the existing ones by implementing an appropriate ExtensionParser and then registering it at the ParserUtils - see the existing ones for details how this can be achieved.

// properietary/special extension parser  
 ParserUtils.registerExtension(new MySpecialExtension());  
  
 try (ClientSession session = client.connect(username, host, port).verify(timeout).getSession()) {  
 session.addPasswordIdentity(password);  
 session.auth().verify(timeout);  
  
 SftpClientFactory factory = SftpClientFactory.instance();  
 try (SftpClient sftp = factory.createSftpClient(session)) {  
 Map<String, byte[]> extensions = sftp.getServerExtensions();  
 // Key=extension name, value=registered parser instance  
 Map<String, ?> data = ParserUtils.parse(extensions);  
 for (Map.Entry<String, ?> de : data.entrySet()) {  
 String extName = de.getKey();  
 Object extValue = de.getValue();  
 if (SftpConstants.EXT\_ACL\_SUPPORTED.equalsIgnoreCase(extName)) {  
 AclCapabilities capabilities = (AclCapabilities) extValue;  
 ...see what other information can be gleaned from it...  
 } else if (SftpConstants.EXT\_VERSIONS.equalsIgnoreCase(extName)) {  
 Versions versions = (Versions) extValue;  
 ...see what other information can be gleaned from it...  
 } else if ("my-special-extension".equalsIgnoreCase(extName)) {  
 MySpecialExtension special = (MySpecialExtension) extValue;  
 ...see what other information can be gleaned from it...  
 } // ...etc....  
 }  
 }  
 }

One can skip all the conditional code if a specific known extension is required:

try (ClientSession session = client.connect(username, host, port).verify(timeout).getSession()) {  
 session.addPasswordIdentity(password);  
 session.auth().verify(timeout);  
  
 SftpClientFactory factory = SftpClientFactory.instance();  
 try (SftpClient sftp = factory.createSftpClient(session)) {  
 // Returns null if extension is not supported by remote server  
 SpaceAvailableExtension space = sftp.getExtension(SpaceAvailableExtension.class);  
 if (space != null) {  
 ...use it...  
 }  
 }  
 }

### Contributing support for a new extension

* Add the code to handle the new extension in AbstractSftpSubsystemHelper#executeExtendedCommand
* Declare the extension name in DEFAULT\_SUPPORTED\_CLIENT\_EXTENSIONS (same class)
* In the org.apache.sshd.sftp.client.extensions.helpers package implement an extension of AbstractSftpClientExtension for sending and receiving the newly added extension.
* Add a relevant parser for reported extension data initial report (if necessary) in ParserUtils#BUILT\_IN\_PARSERS

See how other extensions are implemented and follow their example

## References

* [SFTP drafts for the various versions](https://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/)