

Secure Shell Working Group  
Internet-Draft  
Expires: October 7, 2005

J. Galbraith  
VanDyke Software  
O. Saarenmaa  
F-Secure  
T. Ylonen  
S. Lehtinen  
SSH Communications Security Corp  
April 5, 2005

**SSH File Transfer Protocol**  
**draft-ietf-secsh-filexfer-08.txt**

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of [Section 3 of RFC 3667](#). By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she become aware will be disclosed, in accordance with [RFC 3668](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on October 7, 2005.

Copyright Notice

Copyright (C) The Internet Society (2005).

Abstract

The SSH File Transfer Protocol provides secure file transfer functionality over any reliable data stream. It is the standard file

transfer protocol for use with the SSH2 protocol. This document describes the file transfer protocol and its interface to the SSH2 protocol suite.

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction . . . . .</a>	<a href="#">4</a>
<a href="#">2.</a>	<a href="#">Use with the SSH Connection Protocol . . . . .</a>	<a href="#">4</a>
<a href="#">2.1</a>	<a href="#">The Use of 'stderr' in the server . . . . .</a>	<a href="#">4</a>
<a href="#">3.</a>	<a href="#">General Packet Format . . . . .</a>	<a href="#">5</a>
<a href="#">3.1</a>	<a href="#">Request Synchronization and Reordering . . . . .</a>	<a href="#">6</a>
<a href="#">3.2</a>	<a href="#">New data types defined by this document . . . . .</a>	<a href="#">6</a>
<a href="#">3.3</a>	<a href="#">Packet Types . . . . .</a>	<a href="#">7</a>
<a href="#">4.</a>	<a href="#">Protocol Initialization . . . . .</a>	<a href="#">8</a>
<a href="#">4.1</a>	<a href="#">Client Initialization . . . . .</a>	<a href="#">8</a>
<a href="#">4.2</a>	<a href="#">Server Initialization . . . . .</a>	<a href="#">9</a>
<a href="#">4.3</a>	<a href="#">Determining Server Newline Convention . . . . .</a>	<a href="#">9</a>
<a href="#">4.4</a>	<a href="#">Vendor Id . . . . .</a>	<a href="#">9</a>
<a href="#">4.5</a>	<a href="#">Supported Features . . . . .</a>	<a href="#">10</a>
<a href="#">4.6</a>	<a href="#">Version re-negotiation . . . . .</a>	<a href="#">12</a>
<a href="#">5.</a>	<a href="#">File Names . . . . .</a>	<a href="#">13</a>
<a href="#">6.</a>	<a href="#">File Attributes . . . . .</a>	<a href="#">15</a>
<a href="#">6.1</a>	<a href="#">valid-attribute-flags . . . . .</a>	<a href="#">16</a>
<a href="#">6.2</a>	<a href="#">Type . . . . .</a>	<a href="#">17</a>
<a href="#">6.3</a>	<a href="#">Size . . . . .</a>	<a href="#">17</a>
<a href="#">6.4</a>	<a href="#">allocation-size . . . . .</a>	<a href="#">18</a>
<a href="#">6.5</a>	<a href="#">Owner and Group . . . . .</a>	<a href="#">18</a>
<a href="#">6.6</a>	<a href="#">Permissions . . . . .</a>	<a href="#">19</a>
<a href="#">6.7</a>	<a href="#">Times . . . . .</a>	<a href="#">19</a>
<a href="#">6.8</a>	<a href="#">ACL . . . . .</a>	<a href="#">20</a>
<a href="#">6.9</a>	<a href="#">attrib-bits and attrib-bits-valid . . . . .</a>	<a href="#">21</a>
<a href="#">6.10</a>	<a href="#">text-hint . . . . .</a>	<a href="#">24</a>
<a href="#">6.11</a>	<a href="#">mime-type . . . . .</a>	<a href="#">24</a>
<a href="#">6.12</a>	<a href="#">link-count . . . . .</a>	<a href="#">24</a>
<a href="#">6.13</a>	<a href="#">untranslated-name . . . . .</a>	<a href="#">25</a>
<a href="#">6.14</a>	<a href="#">Extended Attributes . . . . .</a>	<a href="#">25</a>
<a href="#">7.</a>	<a href="#">Requests From the Client to the Server . . . . .</a>	<a href="#">25</a>
<a href="#">7.1</a>	<a href="#">Opening and Closing Files and Directories . . . . .</a>	<a href="#">25</a>
<a href="#">7.1.1</a>	<a href="#">Opening a File . . . . .</a>	<a href="#">26</a>
<a href="#">7.1.2</a>	<a href="#">Opening a Directory . . . . .</a>	<a href="#">31</a>
<a href="#">7.1.3</a>	<a href="#">Closing Handles . . . . .</a>	<a href="#">32</a>
<a href="#">7.2</a>	<a href="#">Reading and Writing . . . . .</a>	<a href="#">32</a>
<a href="#">7.2.1</a>	<a href="#">Reading Files . . . . .</a>	<a href="#">32</a>
<a href="#">7.2.2</a>	<a href="#">Reading Directories . . . . .</a>	<a href="#">33</a>
<a href="#">7.2.3</a>	<a href="#">Writing Files . . . . .</a>	<a href="#">33</a>
<a href="#">7.3</a>	<a href="#">Removing and Renaming Files . . . . .</a>	<a href="#">34</a>
<a href="#">7.4</a>	<a href="#">Creating and Deleting Directories . . . . .</a>	<a href="#">35</a>
<a href="#">7.5</a>	<a href="#">Retrieving File Attributes . . . . .</a>	<a href="#">36</a>



<a href="#">7.6</a>	Setting File Attributes . . . . .	<a href="#">37</a>
<a href="#">7.7</a>	Dealing with Links . . . . .	<a href="#">38</a>
<a href="#">7.8</a>	Canonicalizing the Server-Side Path Name . . . . .	<a href="#">40</a>
<a href="#">7.8.1</a>	Best Practice for Dealing with Paths . . . . .	<a href="#">42</a>
<a href="#">8.</a>	Responses from the Server to the Client . . . . .	<a href="#">42</a>
<a href="#">8.1</a>	Status Response . . . . .	<a href="#">43</a>
<a href="#">8.2</a>	Handle Response . . . . .	<a href="#">47</a>
<a href="#">8.3</a>	Data Response . . . . .	<a href="#">47</a>
<a href="#">8.4</a>	Name Response . . . . .	<a href="#">48</a>
<a href="#">8.5</a>	Attrs Response . . . . .	<a href="#">49</a>
<a href="#">9.</a>	Extensions . . . . .	<a href="#">49</a>
<a href="#">9.1</a>	File Hashing . . . . .	<a href="#">50</a>
<a href="#">9.1.1</a>	Checking File Contents: v5 extension . . . . .	<a href="#">51</a>
<a href="#">9.1.2</a>	Checking File Contents . . . . .	<a href="#">52</a>
<a href="#">9.2</a>	Querying Available Space . . . . .	<a href="#">54</a>
<a href="#">9.3</a>	Querying User Home Directory . . . . .	<a href="#">55</a>
<a href="#">10.</a>	Implementation Considerations . . . . .	<a href="#">55</a>
<a href="#">11.</a>	Security Considerations . . . . .	<a href="#">56</a>
<a href="#">12.</a>	Changes from Previous Protocol Versions . . . . .	<a href="#">57</a>
<a href="#">12.1</a>	Changes Between Versions 6 and 5 . . . . .	<a href="#">57</a>
<a href="#">12.2</a>	Changes Between Versions 5 and 4 . . . . .	<a href="#">58</a>
<a href="#">12.3</a>	Changes Between Versions 4 and 3 . . . . .	<a href="#">59</a>
<a href="#">12.4</a>	Changes Between Versions 3 and 2 . . . . .	<a href="#">59</a>
<a href="#">12.5</a>	Changes Between Versions 2 and 1 . . . . .	<a href="#">59</a>
<a href="#">12.6</a>	Changes Between Versions 1 and 0 . . . . .	<a href="#">59</a>
<a href="#">13.</a>	References . . . . .	<a href="#">60</a>
<a href="#">13.1</a>	Normative References . . . . .	<a href="#">60</a>
<a href="#">13.2</a>	Informative References . . . . .	<a href="#">61</a>
	Authors' Addresses . . . . .	<a href="#">61</a>
	Intellectual Property and Copyright Statements . . . . .	<a href="#">63</a>



## **1. Introduction**

This protocol provides secure file transfer (and more generally file system access.) It is designed so that it could be used to implement a secure remote file system service, as well as a secure file transfer service.

This protocol assumes that it runs over a secure channel, such as a channel in [[I-D.ietf-secsh-architecture](#)], and that the server has already authenticated the client, and that the identity of the client user is available to the protocol.

In general, this protocol follows a simple request-response model. Each request and response contains a sequence number and multiple requests may be pending simultaneously. There are a relatively large number of different request messages, but a small number of possible response messages. Each request has one or more response messages that may be returned in result (e.g., a read either returns data or reports error status).

The packet format descriptions in this specification follow the notation presented in [[I-D.ietf-secsh-architecture](#)].

Even though this protocol is described in the context of the SSH2 protocol, this protocol is general and independent of the rest of the SSH2 protocol suite. It could be used in a number of different applications, such as secure file transfer over TLS [[RFC2246](#)] and transfer of management information in VPN applications.

## **2. Use with the SSH Connection Protocol**

When used with the SSH2 Protocol suite, this protocol is intended to be used as a subsystem as described in [[I-D.ietf-secsh-connect](#)] in the section "Starting a Shell or a Command". The subsystem name used with this protocol is "sftp".

### **2.1 The Use of 'stderr' in the server**

This protocol uses stdout and stdin to transmit binary protocol data. The "session" channel ([[I-D.ietf-secsh-connect](#)]), which is used by the subsystem, also supports the use of stderr.

Data sent on stderr by the server SHOULD be considered free format debug or supplemental error information, and MAY be displayed to the user.

For example, during initialization, there is no client request active, so errors or warning information cannot be sent to the client



as part of the SFTP protocol at this early stage. However, the errors or warnings MAY be sent as stderr text.

### 3. General Packet Format

All packets transmitted over the secure connection are of the following format:

```
uint32      length
byte        type
uint32      request-id
... type specific fields ...
```

#### 'length'

The length of the entire packet, excluding the length field itself, such that, for example, for a packet type containing no type specific fields, the length field would be 5, and 9 bytes of data would be sent on the wire. (This is the packet format used in [\[I-D.ietf-secsh-transport\]](#).)

All packet descriptions in this document omit the length field for brevity; the length field MUST be included in any case.

The maximum size of a packet is in practice determined by the client (the maximum size of read or write requests that it sends, plus a few bytes of packet overhead). All servers SHOULD support packets of at least 34000 bytes (where the packet size refers to the full length, including the header above). This should allow for reads and writes of at most 32768 bytes.

#### 'type'

The type code for the packet.

#### 'request-id'

Each request from the client contains a 'request-id' field. Each response from the server includes that same 'request-id' from the request that the server is responding to. One possible implementation is for the client to use a monotonically increasing request sequence number (modulo  $2^{32}$ ). There is, however, no particular requirement the 'request-id' fields be unique.

There are two packets, INIT and VERSION, which do not use the request-id.

Packet descriptions in this document will contain the 'request-id' field, but will not redefine it.

Implementations MUST ignore excess data at the end of an otherwise





valid packet. Implementations MUST respond to unrecognized packet types with an SSH\_FX\_OP\_UNSUPPORTED error. This will allow the protocol to be extended in a backwards compatible way as needed.

There is no limit on the number of outstanding (non-acknowledged) requests that the client may send to the server. In practice this is limited by the buffering available on the data stream and the queuing performed by the server. If the server's queues are full, it should not read any more data from the stream, and flow control will prevent the client from sending more requests. Note, however, that while there is no restriction on the protocol level, the client's API may provide a limit in order to prevent infinite queuing of outgoing requests at the client.

### **3.1 Request Synchronization and Reordering**

The protocol and implementations MUST process requests relating to the same file in the order in which they are received. In other words, if an application submits multiple requests to the server, the results in the responses will be the same as if it had sent the requests one at a time and waited for the response in each case. For example, the server may process non-overlapping read/write requests to the same file in parallel, but overlapping reads and writes cannot be reordered or parallelized. However, there are no ordering restrictions on the server for processing requests from two different file transfer connections. The server may interleave and parallelize them at will.

There are no restrictions on the order in which responses to outstanding requests are delivered to the client, except that the server must ensure fairness in the sense that processing of no request will be indefinitely delayed even if the client is sending other requests so that there are multiple outstanding requests all the time.

A client MUST be prepared to receive responses to multiple overlapped requests out of order.

### **3.2 New data types defined by this document**

This document defines these data types in addition to those defined in [[I-D.ietf-secsh-architecture](#)].

int64

Represents a 64-bit signed integer. Stored as eight bytes in the order of decreasing significance (network byte order).



#### extension-pair

string extension-name  
string extension-data

'extension-name' is the name of a protocol extension. Extensions not defined by IETF CONSENSUS MUST follow the the DNS extensibility naming convention outlined in [I-D.ietf-secsh-architecture].

'extension-data' is any data specific to the extension, and MAY be zero length if the extension has no data.

### 3.3 Packet Types

The following values are defined for packet types.

SSH_FXP_INIT	1
SSH_FXP_VERSION	2
SSH_FXP_OPEN	3
SSH_FXP_CLOSE	4
SSH_FXP_READ	5
SSH_FXP_WRITE	6
SSH_FXP_LSTAT	7
SSH_FXP_FSTAT	8
SSH_FXP_SETSTAT	9
SSH_FXP_FSETSTAT	10
SSH_FXP_OPENDIR	11
SSH_FXP_READDIR	12
SSH_FXP_REMOVE	13
SSH_FXP_MKDIR	14
SSH_FXP_RMDIR	15
SSH_FXP_REALPATH	16
SSH_FXP_STAT	17
SSH_FXP_RENAME	18
SSH_FXP_READLINK	19
SSH_FXP_LINK	21
SSH_FXP_BLOCK	22
SSH_FXP_UNBLOCK	23
SSH_FXP_STATUS	101
SSH_FXP_HANDLE	102
SSH_FXP_DATA	103
SSH_FXP_NAME	104
SSH_FXP_ATTRS	105
SSH_FXP_EXTENDED	200



## SSH\_FXP\_EXTENDED\_REPLY 201

SSH\_FXP\_EXTENDED and SSH\_FXP\_EXTENDED\_REPLY packets can be used to implement extensions, which can be vendor specific. See Section 'Extensions' for more details.

Values 210-255 are reserved for use in conjunction with these extensions. The SSH\_FXP\_EXTENDED packet can be used to negotiate the meaning of these reserved types. It is suggested that the actual value to be used also be negotiated, since this will prevent collision among multiple uncoordinated extensions.

The server MUST respond with SSH\_FXP\_STATUS(SSH\_FX\_OP\_UNSUPPORTED) if it receives a packet it does not recognize. The protocol version ([Section 4](#)) MUST be incremented if the server is to send new packets to the client (because the client has no way to respond indicating that the packet isn't recognized.)

## [4.](#) Protocol Initialization

When the file transfer protocol starts, the client first sends a SSH\_FXP\_INIT (including its version number) packet to the server. The server responds with a SSH\_FXP\_VERSION packet, supplying the lowest of its own and the client's version number. Both parties should from then on adhere to that particular version of the protocol.

The version number of the protocol specified in this document is 6. The version number should be incremented for each incompatible revision of this protocol.

Note that these two packets DO NOT contain a request id. These are the only such packets in the protocol.

### [4.1](#) Client Initialization

The SSH\_FXP\_INIT packet (from client to server) has the following data:

uint32 version

'version' is the version number of the client. If the client wishes to interoperate with servers that support discontinuous version numbers it SHOULD send '3', and then use the 'version-select' extension (see below.) Otherwise, this value is '6' for this version of the protocol.



## **[4.2](#) Server Initialization**

The SSH\_FXP\_VERSION packet (from server to client) has the following data:

```
uint32 version
extension-pair extensions[0..n]
```

'version' is the lower of the protocol version supported by the server and the version number received from the client.

'extensions' is 0 or more extension-pairs ([Section 3.2](#)). Implementations MUST silently ignore any extensions whose names they do not recognize.

## **[4.3](#) Determining Server Newline Convention**

In order to correctly process text files in a cross platform compatible way, newline sequences must be converted between client and server conventions.

The SSH\_FXP\_TEXT file open flag ([Section 7.1.1](#)) makes it possible to request that the server translate a file to a 'canonical' wire format. This format uses CRLF as the line separator.

Servers for systems using other conventions MUST translate to and from the canonical form.

However, to ease the burden of implementation on servers that use a single, simple, separator sequence the following extension allows the canonical format to be changed.

```
string "newline"
string new-canonical-separator (usually CR or LF or CRLF)
```

All clients MUST support this extension.

When processing text files, clients SHOULD NOT translate any character or sequence that is not an exact match of the server's newline separator.

In particular, if the newline sequence being used is the canonical CRLF sequence, a lone CR or a lone LF SHOULD be written through without change.

## **[4.4](#) Vendor Id**

It is often necessary to detect the version of a the server so that





bugs can be worked around. This extension allows the client to do so. (It may also be sent by the client using an EXTENDED request.

```
string "vendor-id"  
string vendor-structure  
    string vendor-name  
    string product-name  
    string product-version  
    uint64 product-build-number
```

vendor-name

Arbitrary name identifying the maker of the product.

product-name

Arbitrary name identifying the product.

product-name

Arbitrary string identifying the version of the product.

product-build-number

A build-number for the product, such that if a bug is fixed in build-number 'x', it can be assumed that (barring regression in the product) it is fixed in all build-numbers > 'x'.

#### [4.5](#) Supported Features

The sftp protocol has grown to be very rich, and now supports a number of features that may not be available on all servers.

When a server receives a request for a feature it cannot support, it MUST return a SSH\_FX\_OP\_UNSUPPORTED status code, unless otherwise specified. The following extension facilitates clients being able to use the maximum available feature set, and yet not be overly burdened by dealing with SSH\_FX\_OP\_UNSUPPORTED status codes. All servers MUST include as part of their version packet.

```
string "supported2"  
string supported-structure  
    uint32 supported-attribute-mask  
    uint32 supported-attribute-bits  
    uint32 supported-open-flags  
    uint32 max-read-size  
    uint64 supported-open-block-masks  
    uint64 supported-block-masks  
    uint32 attrib-extension-count  
    string attrib-extension-names[attrib_extension-count]  
    uint32 extension-count
```



string extension-names[extension-count]

Note that the name "supported2" is used here to avoid conflict with the slightly different "supported" extension that was previously used.

supported-attribute-mask

This mask MAY be applied to the 'File Attributes' valid-attribute-flags field ([Section 6.1](#)) to ensure that no unsupported attributes are present during a operation which writes attributes.

supported-attribute-bits

This mask MAY be applied to the 'File Attributes' attrib-bits field ([Section 6.9](#)) to ensure that no unsupported attrib-bits are present during a operation which writes attributes.

supported-open-flags

The supported-open-flags mask MAY be applied to the SSH\_FXP\_OPEN ([Section 7.1.1](#)) flags field.

max-read-size

This is the maximum read size that the server guarantees to complete. For example, certain embedded server implementations complete only the first 4K of a read, even if there is additional data to be read from the file.

If the server specifies a non-zero value for max-read-size, it MUST return the requested number of bytes for reads that are less than or equal to the value, unless it encounters EOF or an ERROR.

The server MAY use this value to express that it is willing to handle very large read requests, in excess of the standard 34000 bytes specified in [Section 3](#).

supported-open-block-masks

Series of four-bit fields representing the support combinations of SSH\_FXF\_ACCESS\_BLOCK\_READ, SSH\_FXF\_ACCESS\_BLOCK\_WRITE, SSH\_FXF\_ACCESS\_BLOCK\_DELETE, and SSH\_FXF\_ACCESS\_BLOCK\_ADVISORY ([Section 7.1.1.3](#).)

The bits are interpreted in four-bit groups, beginning with the least significant bit. The bit values are the values used in the file open flags, shifted right so that BLOCK\_READ is the least significant bit.



For example, a server that supported only the classic advisory read lock (shared lock) and write lock (exclusive lock) would send 0b1011 1010, or 0xba.

#### supported-block-masks

Series of four-bit fields representing the supported combinations of BLOCK\_\* flags for use with the SSH\_FXF\_BLOCK request, as described above.

#### attrib-extension-count

Count of extension names in the attrib-extension array.

#### attrib-extension-names

Names of extensions that can be used in an ATTRS ([Section 6.14](#)) structure.

#### extension-count

Count of extension names in the attrib-extension array.

#### extension-names

Names of extensions that can be used with the SSH\_FXP\_EXTEND ([Section 9](#)) packet.

Naturally, if a given attribute field, attribute mask bit, open flag, or extension is required for correct operation, the client **MUST** either not allow the bit to be masked off, or **MUST** fail the operation gracefully without sending the request to the server.

The client **MAY** send requests that are not supported by the server; however, it is not normally expected to be productive to do so. The client **SHOULD** apply the mask even to attrib structures received from the server. The server **MAY** include attributes or attrib-bits that are not included in the mask. Such attributes or attrib-bits are effectively read-only.

## [4.6](#) Version re-negotiation

If the server supports other versions than what was negotiated, it may wish to send the 'versions' extension to inform the client of this fact. The client may then optionally choose to use one of the other versions supported.

```
string "versions"  
string comma-separated-versions
```

'comma-separated-versions' is a string of comma separated version numbers, for example, "3,6,7". Defined versions are: "2", "3", "4", "5", "6". Any other version advertised by the server must follow the



DNS extensibility naming convention outlined in [I-D.ietf-secsh-architecture].

If the client and server have negotiated any version higher than version '3' (the version at which SSH\_FXP\_EXTENDED was introduced) in the initial VERSION/INIT exchange, the client may select a new version to use from the list the server provided using the following SSH\_FXP\_EXTENDED request.

```
string "version-select"  
uint32 version-from-list
```

If the 'version-from-list' is one of the versions on the servers list, the server MUST respond with SSH\_FX\_OK. If the server did not send the "versions" extension, or the version-from-list was not included, the server MAY send a status response describing the failure, but MUST then close the channel without processing any further requests.

The 'version-select' MUST be the first request from the client to the server; if it is not, the server MUST fail the request and close the channel.

Although this request does take a full round trip, no client need wait for the response before continuing, because any valid request MUST succeed, and any invalid request results in a channel close. Since the request is the first request, it is not possible for the server to have already sent responses conforming to the old version.

The client SHOULD NOT select a version lower than was initially negotiated; however, it is not forbidden to do so. One reason a client might do so is to work around a buggy implementation.

## **5. File Names**

This protocol represents file names as strings. File names are assumed to use the slash ('/') character as a directory separator.

File names starting with a slash are "absolute", and are relative to the root of the file system. Names starting with any other character are relative to the user's default directory (home directory). Note that identifying the user is assumed to take place outside of this protocol.

Servers SHOULD interpret a path name component ".." ([Section 11](#)) as referring to the parent directory, and "." as referring to the current directory.





An empty path name is valid, and it refers to the user's default directory (usually the user's home directory).

Otherwise, no syntax is defined for file names by this specification. Clients should not make any other assumptions; however, they can splice path name components returned by SSH\_FXP\_READDIR together using a slash ('/') as the separator, and that will work as expected.

It is understood that the lack of well-defined semantics for file names may cause interoperability problems between clients and servers using radically different operating systems. However, this approach is known to work acceptably with most systems, and alternative approaches that e.g. treat file names as sequences of structured components are quite complicated.

The preferred encoding for filenames is UTF-8. This is consistent with IETF Policy on Character Sets and Languages [[RFC2277](#)] and it is further supposed that the server is more likely to support any local character set and be able to convert it to UTF-8.

However, because the server does not always know the encoding of filenames, it is not always possible for the server to perform a valid translation to UTF-8. When an invalid translation to UTF-8 is performed, it becomes impossible to manipulate the file, because the translation is not reversible. Therefore, the following extensions are provided in order to make it possible for the server to communicate its abilities to the client, and to allow the client to control whether the server attempts the conversion.

A server MAY include the following extension with its version packet.

```
string "filename-charset"  
string charset-name
```

A server that can always provide a valid UTF-8 translation for filenames SHOULD NOT send this extension. Otherwise, the server SHOULD send this extension and include the encoding most likely to be used for filenames. This value will most likely be derived from the LC\_CTYPE on most unix-like systems.

A server that does not send this extension MUST send all filenames encoded in UTF-8. All clients MUST support UTF-8 filenames.

If the server included the 'filename-charset' extension with its VERSION packet, a client MAY send the following extension to turn off server translation to UTF-8.



```
string  "filename-translation-control"  
bool    do-translate
```

If the client does not send this extension, the server MUST continue to attempt translation to UTF-8. When a client sends this extension, the server MUST enable filename translation if 'do-translate' is true, or disable filename translation if it is false.

The server MUST respond with a STATUS response; if the server sent a 'filename-charset' extension, the status MUST be SUCCESS. Otherwise, the status MUST be UNSUPPORTED.

When UTF-8 is sent, the shortest valid UTF-8 encoding of the UNICODE data MUST be used. The server is responsible for converting the UNICODE data to whatever canonical form it requires. For example, if the server requires that precomposed characters always be used, the server MUST NOT assume the filename as sent by the client has this attribute, but must do this normalization itself.

## 6. File Attributes

A new compound data type, 'ATTRS', is defined for encoding file attributes. The same encoding is used both when returning file attributes from the server and when sending file attributes to the server.



uint32	valid-attribute-flags	
byte	type	always present
uint64	size	if flag SIZE
uint64	allocation-size	if flag ALLOCATION_SIZE
string	owner	if flag OWNERGROUP
string	group	if flag OWNERGROUP
uint32	permissions	if flag PERMISSIONS
int64	atime	if flag ACESSTIME
uint32	atime-nseconds	if flag SUBSECOND_TIMES
int64	createtime	if flag CREATETIME
uint32	createtime-nseconds	if flag SUBSECOND_TIMES
int64	mtime	if flag MODIFYTIME
uint32	mtime-nseconds	if flag SUBSECOND_TIMES
int64	ctime	if flag CTIME
uint32	ctime-nseconds	if flag SUBSECOND_TIMES
string	acl	if flag ACL
uint32	attrib-bits	if flag BITS
uint32	attrib-bits-valid	if flag BITS
byte	text-hint	if flag TEXT_HINT
string	mime-type	if flag MIME_TYPE
uint32	link-count	if flag LINK_COUNT
string	untranslated-name	if flag UNTRANSLATED_NAME
uint32	extended-count	if flag EXTENDED
extended-pair extensions		

### 6.1 valid-attribute-flags

The 'valid-attribute-flags' specifies which of the fields are present. Those fields for which the corresponding flag is not set are not present (not included in the packet).

The server generally includes all attributes it knows about; however, it may exclude attributes that are overly expensive to retrieve unless the client explicitly requests them.

When writing attributes, the server SHOULD NOT modify attributes that are not present in the structure. However, if necessary, the server MAY use a default value for an absent attribute.

In general, unless otherwise specified, if a server cannot support writing an attribute requested, it must fail the setstat operation. In this case, none of the attributes SHOULD be changed.

New fields can only be added by incrementing the protocol version number (or by using the extension mechanism described below).

The following values are defined:



SSH_FILEXFER_ATTR_SIZE	0x00000001
SSH_FILEXFER_ATTR_PERMISSIONS	0x00000004
SSH_FILEXFER_ATTR_ACCESSTIME	0x00000008
SSH_FILEXFER_ATTR_CREATETIME	0x00000010
SSH_FILEXFER_ATTR_MODIFYTIME	0x00000020
SSH_FILEXFER_ATTR_ACL	0x00000040
SSH_FILEXFER_ATTR_OWNERGROUP	0x00000080
SSH_FILEXFER_ATTR_SUBSECOND_TIMES	0x00000100
SSH_FILEXFER_ATTR_BITS	0x00000200
SSH_FILEXFER_ATTR_ALLOCATION_SIZE	0x00000400
SSH_FILEXFER_ATTR_TEXT_HINT	0x00000800
SSH_FILEXFER_ATTR_MIME_TYPE	0x00001000
SSH_FILEXFER_ATTR_LINK_COUNT	0x00002000
SSH_FILEXFER_ATTR_UNTRANSLATED_NAME	0x00004000
SSH_FILEXFER_ATTR_CTIME	0x00008000
SSH_FILEXFER_ATTR_EXTENDED	0x80000000

0x00000002 was used in a previous version of this protocol. It is now a reserved value and MUST NOT appear in the mask. Some future version of this protocol may reuse this value.

## 6.2 Type

The type field is always present. The following types are defined:

SSH_FILEXFER_TYPE_REGULAR	1
SSH_FILEXFER_TYPE_DIRECTORY	2
SSH_FILEXFER_TYPE_SYMLINK	3
SSH_FILEXFER_TYPE_SPECIAL	4
SSH_FILEXFER_TYPE_UNKNOWN	5
SSH_FILEXFER_TYPE_SOCKET	6
SSH_FILEXFER_TYPE_CHAR_DEVICE	7
SSH_FILEXFER_TYPE_BLOCK_DEVICE	8
SSH_FILEXFER_TYPE_FIFO	9

On a POSIX system, these values would be derived from the mode field of the stat structure. SPECIAL should be used for files that are of a known type which cannot be expressed in the protocol. UNKNOWN should be used if the type is not known.

## 6.3 Size

The 'size' field specifies the number of bytes that can be read from the file, or in other words, the location of the end-of-file. This attribute MUST NOT be present during file creation.

If this field is present during a setstat operation, the file MUST be extended or truncated to the specified size.





Files opened with the `SSH_FXF_ACCESS_TEXT` flag may have a size that is greater or less than the value of the size field. The server MAY fail `setstat` operations specifying size for files opened with the `SSH_FXF_ACCESS_TEXT` flag.

#### **6.4 allocation-size**

The 'allocation-size' field specifies the number of bytes that the file consumes on disk. This field MAY be less than the 'size' field if the file is 'sparse' ([Section 6.9](#)).

When present during file creation, the file SHOULD be created and the specified number of bytes pre-allocated. If the pre-allocation fails, the file should be removed (if it was created) and an error returned.

If this field is present during a `setstat` operation, the file SHOULD be extended or truncated to the specified size. The 'size' of the file may be affected by this operation. If the operation succeeds, the 'size' should be the minimum of the 'size' before the operation and the new 'allocation-size'.

Querying the 'allocation-size' after setting it MUST return a value that is greater-than or equal to the value set, but it MAY not return the precise value set.

If both 'size' and 'allocation-size' are set during a `setstat` operation, and 'allocation-size' is less than 'size', the server MUST return `SSH_FX_INVALID_PARAMETER`.

#### **6.5 Owner and Group**

The 'owner' and 'group' fields are represented as UTF-8 strings; this is the form used by NFS v4. See NFS version 4 Protocol [[RFC3010](#)]. The following text is selected quotations from [section 5.6](#).

To avoid a representation that is tied to a particular underlying implementation at the client or server, the use of UTF-8 strings has been chosen. The string should be of the form "user@dns\_domain". This will allow for a client and server that do not use the same local representation the ability to translate to a common syntax that can be interpreted by both. In the case where there is no translation available to the client or server, the attribute value must be constructed without the "@". Therefore, the absence of the @ from the owner or owner\_group attribute signifies that no translation was available and the receiver of the attribute should not place any special meaning on the attribute value. Even though the attribute value cannot be translated, it may still be useful. In the case of a



client, the attribute string may be used for local display of ownership.

user@localhost represents a user in the context of the server.

If either the owner or group field is zero length, the field should be considered absent, and no change should be made to that specific field during a modification operation.

## 6.6 Permissions

The 'permissions' field contains a bit mask specifying file permissions. These permissions correspond to the st\_mode field of the stat structure defined by POSIX [[IEEE.1003-1.1996](#)].

This protocol uses the following values for the symbols declared in the POSIX standard.

S_IRUSR	0000400 (octal)
S_IWUSR	0000200
S_IXUSR	0000100
S_IRGRP	0000040
S_IWGRP	0000020
S_IXGRP	0000010
S_IROTH	0000004
S_IWOTH	0000002
S_IXOTH	0000001
S_ISUID	0004000
S_ISGID	0002000
S_ISVTX	0001000

Implementations MUST NOT send bits that are not defined.

The server SHOULD NOT apply a 'umask' to the mode bits; but should set the mode bits as specified by the client. The client MUST apply an appropriate 'umask' to the mode bits before sending them.

## 6.7 Times

The 'atime' field contains the last access time of the file. Many operating systems either don't have this field, only optionally maintain it, or maintain it with less resolution than other fields.

The 'mtime' contains the last time the file was written.

'createtime' contains the creation time of the file.

'ctime' contains the last time the file attributes were changed. The



exact meaning of this field depends on the server.

All times are represented as seconds from Jan 1, 1970 in UTC. A negative value indicates number of seconds before Jan 1, 1970. In both cases, if the SSH\_FILEXFER\_ATTR\_SUBSECOND\_TIMES flag is set, the nseconds field is to be added to the seconds field for the final time representation. For example, if the time to be represented is one-half second before 0 hour January 1, 1970, the seconds field would have a value of negative one (-1) and the nseconds fields would have a value of one-half second (500000000). Values greater than 999,999,999 for nseconds are considered invalid.

## 6.8 ACL

The 'ACL' field contains an ACL similar to that defined in [section 5.9](#) of NFS version 4 Protocol [[RFC3010](#)].

```
uint32    ace-count

repeated ace-count time:
uint32    ace-type
uint32    ace-flag
uint32    ace-mask
string    who [UTF-8]
```

ace-type is one of the following four values (taken from NFS Version 4 Protocol [[RFC3010](#)]):

```
ACE4_ACCESS_ALLOWED_ACE_TYPE 0x00000000
ACE4_ACCESS_DENIED_ACE_TYPE  0x00000001
ACE4_SYSTEM_AUDIT_ACE_TYPE   0x00000002
ACE4_SYSTEM_ALARM_ACE_TYPE   0x00000003
```

ace-flag is a combination of the following flag values. See NFS Version 4 Protocol [[RFC3010](#)] [section 5.9.2](#):

```
ACE4_FILE_INHERIT_ACE          0x00000001
ACE4_DIRECTORY_INHERIT_ACE     0x00000002
ACE4_NO_PROPAGATE_INHERIT_ACE  0x00000004
ACE4_INHERIT_ONLY_ACE          0x00000008
ACE4_SUCCESSFUL_ACCESS_ACE_FLAG 0x00000010
ACE4_FAILED_ACCESS_ACE_FLAG    0x00000020
ACE4_IDENTIFIER_GROUP          0x00000040
```

ace-mask is any combination of the following flags (taken from [[RFC3010](#)], [section 5.9.3](#)). The semantic meaning of these flags is also given in [[RFC3010](#)].



ACE4_READ_DATA	0x00000001
ACE4_LIST_DIRECTORY	0x00000001
ACE4_WRITE_DATA	0x00000002
ACE4_ADD_FILE	0x00000002
ACE4_APPEND_DATA	0x00000004
ACE4_ADD_SUBDIRECTORY	0x00000004
ACE4_READ_NAMED_ATTRS	0x00000008
ACE4_WRITE_NAMED_ATTRS	0x00000010
ACE4_EXECUTE	0x00000020
ACE4_DELETE_CHILD	0x00000040
ACE4_READ_ATTRIBUTES	0x00000080
ACE4_WRITE_ATTRIBUTES	0x00000100
ACE4_DELETE	0x00010000
ACE4_READ_ACL	0x00020000
ACE4_WRITE_ACL	0x00040000
ACE4_WRITE_OWNER	0x00080000
ACE4_SYNCHRONIZE	0x00100000

who is a UTF-8 string of the form described in 'Owner and Group' ([Section 6.5](#))

Also, as per '5.9.4 ACE who' [[RFC3010](#)] there are several identifiers that need to be understood universally. Some of these identifiers cannot be understood when an client access the server, but have meaning when a local process accesses the file. The ability to display and modify these permissions is permitted over SFTP.

OWNER	The owner of the file.
GROUP	The group associated with the file.
EVERYONE	The world.
INTERACTIVE	Accessed from an interactive terminal.
NETWORK	Accessed via the network.
DIALUP	Accessed as a dialup user to the server.
BATCH	Accessed from a batch job.
ANONYMOUS	Accessed without any authentication.
AUTHENTICATED	Any authenticated user (opposite of ANONYMOUS).
SERVICE	Access from a system service.

To avoid conflict, these special identifiers are distinguish by an appended "@". For example: ANONYMOUS@.

## [6.9](#) attrib-bits and attrib-bits-valid

These fields, taken together, reflect various attributes of the file or directory, on the server.

Bits not set in 'attrib-bits-valid' MUST be ignored in the 'attrib-bits' field. This allows both the server and the client to





communicate only the bits it knows about without inadvertently twiddling bits they don't understand.

The following attrib-bits are defined:

SSH_FILEXFER_ATTR_FLAGS_READONLY	0x00000001
SSH_FILEXFER_ATTR_FLAGS_SYSTEM	0x00000002
SSH_FILEXFER_ATTR_FLAGS_HIDDEN	0x00000004
SSH_FILEXFER_ATTR_FLAGS_CASE_INSENSITIVE	0x00000008
SSH_FILEXFER_ATTR_FLAGS_ARCHIVE	0x00000010
SSH_FILEXFER_ATTR_FLAGS_ENCRYPTED	0x00000020
SSH_FILEXFER_ATTR_FLAGS_COMPRESSED	0x00000040
SSH_FILEXFER_ATTR_FLAGS_SPARSE	0x00000080
SSH_FILEXFER_ATTR_FLAGS_APPEND_ONLY	0x00000100
SSH_FILEXFER_ATTR_FLAGS_IMMUTABLE	0x00000200
SSH_FILEXFER_ATTR_FLAGS_SYNC	0x00000400
SSH_FILEXFER_ATTR_FLAGS_TRANSLATION_ERR	0x00000800

#### SSH\_FILEXFER\_ATTR\_FLAGS\_READONLY

Advisory, read-only bit. This bit is not part of the access control information on the file, but is rather an advisory field indicating that the file should not be written.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_SYSTEM

The file is part of the operating system.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_HIDDEN

File SHOULD NOT be shown to user unless specifically requested. For example, most UNIX systems SHOULD set this bit if the filename begins with a 'period'. This bit may be read-only ([Section 4.5](#)). Most UNIX systems will not allow this to be changed.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_CASE\_INSENSITIVE

This attribute applies only to directories. This attribute is always read-only, and cannot be modified. This attribute means that files and directory names in this directory should be compared without regard to case.

It is recommended that where possible, the server's filesystem be allowed to do comparisons. For example, if a client wished to prompt a user before overwriting a file, it should not compare the new name with the previously retrieved list of names in the directory. Rather, it should first try to create the new file by specifying SSH\_FXF\_CREATE\_NEW flag. Then, if this fails and returns SSH\_FX\_FILE\_ALREADY\_EXISTS, it should prompt the user and then retry the create specifying SSH\_FXF\_CREATE\_TRUNCATE.



Unless otherwise specified, filenames are assumed to be case sensitive.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_ARCHIVE

The file should be included in backup / archive operations.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_ENCRYPTED

The file is stored on disk using file-system level transparent encryption. This flag does not affect the file data on the wire (for either READ or WRITE requests.)

#### SSH\_FILEXFER\_ATTR\_FLAGS\_COMPRESSED

The file is stored on disk using file-system level transparent compression. This flag does not affect the file data on the wire.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_SPARSE

The file is a sparse file; this means that file blocks that have not been explicitly written are not stored on disk. For example, if a client writes a buffer at 10 M from the beginning of the file, the blocks between the previous EOF marker and the 10 M offset would not consume physical disk space.

Some servers may store all files as sparse files, in which case this bit will be unconditionally set. Other servers may not have a mechanism for determining if the file is sparse, and so the file MAY be stored sparse even if this flag is not set.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_APPEND\_ONLY

Opening the file without either the SSH\_FXF\_ACCESS\_APPEND\_DATA or the SSH\_FXF\_ACCESS\_APPEND\_DATA\_ATOMIC flag ([Section 7.1.1.3](#)) MUST result in an SSH\_FX\_INVALID\_PARAMETER error.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_IMMUTABLE

The file cannot be deleted or renamed, no hard link can be created to this file, and no data can be written to the file.

This bit implies a stronger level of protection than SSH\_FILEXFER\_ATTR\_FLAGS\_READONLY, the file permission mask or ACLs. Typically even the superuser cannot write to immutable files, and only the superuser can set or remove the bit.

#### SSH\_FILEXFER\_ATTR\_FLAGS\_SYNC

When the file is modified, the changes are written synchronously to the disk.



**SSH\_FILEXFER\_ATTR\_FLAGS\_TRANSLATION\_ERR**

The server MAY include this bit in a directory listing or realpath response. It indicates there was a failure in the translation to UTF-8. If this flag is included, the server SHOULD also include the UNTRANSLATED\_NAME attribute.

**6.10 text-hint**

The value is one of the following enumerations, and indicates what the server knows about the content of the file.

SSH_FILEXFER_ATTR_KNOWN_TEXT	0x00
SSH_FILEXFER_ATTR_GUESSED_TEXT	0x01
SSH_FILEXFER_ATTR_KNOWN_BINARY	0x02
SSH_FILEXFER_ATTR_GUESSED_BINARY	0x03

**SSH\_FILEXFER\_ATTR\_KNOWN\_TEXT**

The server knows the file is a text file, and should be opened using the SSH\_FXF\_ACCESS\_TEXT\_MODE flag.

**SSH\_FILEXFER\_ATTR\_GUESSED\_TEXT**

The server has applied a heuristic or other mechanism and believes that the file should be opened with the SSH\_FXF\_ACCESS\_TEXT\_MODE flag.

**SSH\_FILEXFER\_ATTR\_KNOWN\_BINARY**

The server knows the file has binary content.

**SSH\_FILEXFER\_ATTR\_GUESSED\_BINARY**

The server has applied a heuristic or other mechanism and believes has binary content, and should not be opened with the SSH\_FXF\_ACCESS\_TEXT\_MODE flag.

This flag MUST NOT be present during either a setstat or a fsetstat operation.

**6.11 mime-type**

The 'mime-type' field contains the mime-type [[RFC1521](#)] string. Most servers will not know this information and should not set the bit in their supported-attribute-mask.

**6.12 link-count**

This field contains the hard link count of the file. This attribute MUST NOT be present during a setstat operation.



### **6.13 untranslated-name**

This field contains the name before filename translation was attempted. It MUST NOT be included unless the server also set the SSH\_FILEXFER\_ATTR\_FLAGS\_TRANSLATION\_ERR ([Section 6.9](#)) bit in the attrib-bits field.

### **6.14 Extended Attributes**

The SSH\_FILEXFER\_ATTR\_EXTENDED flag provides a general extension mechanism for the attrib structure. If the flag is specified, then the 'extended\_count' field is present. It specifies the number of 'extension-pair' items that follow. Each of these items specifies an extended attribute. Implementations MUST return SSH\_FX\_UNSUPPORTED if there are any unrecognized extensions. Clients can avoid sending unsupported extensions by examining the attrib-extension-names of the "supported2" extension attrib-extension-names ([Section 4.5](#)).

Additional fields can be added to the attributes by either defining additional bits to the flags field to indicate their presence, or by defining extended attributes for them. The extended attributes mechanism is recommended for most purposes; additional flags bits should only be defined by an IETF standards action that also increments the protocol version number. The use of such new fields MUST be negotiated by the version number in the protocol exchange. It is a protocol error if a packet with unsupported protocol bits is received.

## **7. Requests From the Client to the Server**

Requests from the client to the server represent the various file system operations.

### **7.1 Opening and Closing Files and Directories**

Many operations in the protocol operate on open files. The SSH\_FXP\_OPEN and SSH\_FXP\_OPENDIR requests return a handle (which is an opaque, variable-length string) which may be used to access the file or directory later. The client MUST NOT send requests to the server with bogus or closed handles. However, the server MUST perform adequate checks on the handle in order to avoid security risks due to fabricated handles.

This design allows either stateful and stateless server implementation, as well as an implementation which caches state between requests but may also flush it. The contents of the file handle string are entirely up to the server and its design. The client should not modify or attempt to interpret the file handle





strings.

The file handle strings MUST NOT be longer than 256 bytes.

#### **7.1.1 Opening a File**

Files are opened and created using the SSH\_FXP\_OPEN message.

```
byte    SSH_FXP_OPEN
uint32  request-id
string  filename [UTF-8]
uint32  desired-access
uint32  flags
ATTRS   attrs
```

The response to this message will be either SSH\_FXP\_HANDLE (if the operation is successful) or SSH\_FXP\_STATUS (if the operation fails.)

##### **7.1.1.1 filename**

The 'filename' field specifies the file name. See Section 'File Names' for more information. If 'filename' is a directory file, the server MUST return an SSH\_FX\_FILE\_IS\_A\_DIRECTORY error.

##### **7.1.1.2 desired-access**

The 'desired-access' field is a bitmask containing a combination of values from the ace-mask flags ([Section 6.8](#)). Note that again, the meaning of these flags is given in [[RFC3010](#)].

The server MUST be prepared to translate the SFTP access flags into its local equivalents. If the server cannot grant the access desired, it MUST return SSH\_FX\_PERMISSION\_DENIED.

The server MAY open the file with greater access than requested if the user has such access and the server implementation requires it. For example, a server that does not distinguish between READ\_ATTRIBUTE and READ\_DATA will have to request full 'read' access to the file when the client only requested READ\_ATTRIBUTE, resulting in greater access than the client originally requested.

In such cases, it is possible, and permissible in the protocol, that the client could open a file requesting some limited access, and then access the file in a way not permitted by that limited access and the server would permit such action. However, the server MUST NOT ever grant access to the file that the client does not actually have the rights to.



### 7.1.1.3 flags

The 'flags' field controls various aspects of the operation, including whether or not the file is created and the kind of locking desired.

The following 'flags' are defined:

SSH_FXF_ACCESS_DISPOSITION	= 0x00000007
SSH_FXF_CREATE_NEW	= 0x00000000
SSH_FXF_CREATE_TRUNCATE	= 0x00000001
SSH_FXF_OPEN_EXISTING	= 0x00000002
SSH_FXF_OPEN_OR_CREATE	= 0x00000003
SSH_FXF_TRUNCATE_EXISTING	= 0x00000004
SSH_FXF_ACCESS_APPEND_DATA	= 0x00000008
SSH_FXF_ACCESS_APPEND_DATA_ATOMIC	= 0x00000010
SSH_FXF_ACCESS_TEXT_MODE	= 0x00000020
SSH_FXF_ACCESS_BLOCK_READ	= 0x00000040
SSH_FXF_ACCESS_BLOCK_WRITE	= 0x00000080
SSH_FXF_ACCESS_BLOCK_DELETE	= 0x00000100
SSH_FXF_ACCESS_BLOCK_ADVISORY	= 0x00000200
SSH_FXF_ACCESS_NOFOLLOW	= 0x00000400
SSH_FXF_ACCESS_DELETE_ON_CLOSE	= 0x00000800

#### SSH\_FXF\_ACCESS\_DISPOSITION

Disposition is a 3 bit field that controls how the file is opened. The server MUST support these bits. Any one of the following enumeration is allowed:

#### SSH\_FXF\_CREATE\_NEW

A new file is created; if the file already exists, the server MUST return status SSH\_FX\_FILE\_ALREADY\_EXISTS.

#### SSH\_FXF\_CREATE\_TRUNCATE

A new file is created; if the file already exists, it is opened and truncated.

#### SSH\_FXF\_OPEN\_EXISTING

An existing file is opened. If the file does not exist, the server MUST return SSH\_FX\_NO\_SUCH\_FILE. If a directory in the path does not exist, the server SHOULD return SSH\_FX\_NO\_SUCH\_PATH. It is also acceptable if the server returns SSH\_FX\_NO\_SUCH\_FILE in this case.



**SSH\_FXF\_OPEN\_OR\_CREATE**

If the file exists, it is opened. If the file does not exist, it is created.

**SSH\_FXF\_TRUNCATE\_EXISTING**

An existing file is opened and truncated. If the file does not exist, the server **MUST** return the same error codes as defined for **SSH\_FXF\_OPEN\_EXISTING**.

**SSH\_FXF\_ACCESS\_APPEND\_DATA**

Data is always written at the end of the file. The offset field of the **SSH\_FXP\_WRITE** requests are ignored.

Data is not required to be appended atomically. This means that if multiple writers attempt to append data simultaneously, data from the first may be lost. However, data **MAY** be appended atomically.

**SSH\_FXF\_ACCESS\_APPEND\_DATA\_ATOMIC**

Data is always written at the end of the file. The offset field of the **SSH\_FXP\_WRITE** requests are ignored.

Data **MUST** be written atomically so that there is no chance that multiple appenders can collide and result in data being lost.

If both append flags are specified, the server **SHOULD** use atomic append if it is available, but **SHOULD** use non-atomic appends otherwise. The server **SHOULD NOT** fail the request in this case.

**SSH\_FXF\_TEXT**

Indicates that the server should treat the file as text and convert it to the canonical newline convention in use. (See Determining Server Newline Convention. ([Section 4.3](#)))

When a file is opened with the **FXF\_TEXT** flag, the offset field in the read and write functions is ignored.

Servers **MUST** process multiple, parallel reads and writes correctly in this mode. Naturally, it is permissible for them to do this by serializing the requests.

Clients **SHOULD** use the **SSH\_FXF\_ACCESS\_APPEND\_DATA** flag to append data to a text file rather than using write with a calculated offset.



To support seeks on text files the following SSH\_FXP\_EXTENDED packet is defined.

```
string "text-seek"  
string file-handle  
uint64 line-number
```

line-number is the index of the line number to seek to, where byte 0 in the file is line number 0, and the byte directly following the first newline sequence in the file is line number 1 and so on.

The response to a "text-seek" request is an SSH\_FXP\_STATUS message.

An attempt to seek past the end-of-file should result in a SSH\_FX\_EOF status.

Servers SHOULD support at least one "text-seek" in order to support resume. However, a client MUST be prepared to receive SSH\_FX\_OP\_UNSUPPORTED when attempting a "text-seek" operation. The client can then try a fall-back strategy, if it has one.

Clients MUST be prepared to handle SSH\_FX\_OP\_UNSUPPORTED returned for read or write operations that are not sequential.

#### SSH\_FXF\_ACCESS\_BLOCK\_READ

The server MUST guarantee that no other open has taken place which blocked handle has been opened with ACE4\_READ\_DATA access, and that no other handle will be opened with ACE4\_READ\_DATA access until the client closes the handle. (This MUST apply both to other clients and to other processes on the server.)

If there is a conflicting lock the server MUST return SSH\_FX\_LOCK\_CONFLICT. If the server cannot make the locking guarantee, it MUST return SSH\_FX\_OP\_UNSUPPORTED.

Other handles MAY be opened for ACE4\_WRITE\_DATA or any other combination of accesses, as long as ACE4\_READ\_DATA is not included in the mask.

#### SSH\_FXF\_ACCESS\_BLOCK\_WRITE

The server MUST guarantee that no other lock has been opened with ACE4\_WRITE\_DATA or ACE4\_APPEND\_DATA access, and that no other handle will be opened with ACE4\_WRITE\_DATA or ACE4\_APPEND\_DATA access until the client closes the handle. (This MUST apply both to other clients and to other processes on the server.)





If there is a conflicting lock the server MUST return SSH\_FX\_LOCK\_CONFLICT. If the server cannot make the locking guarantee, it MUST return SSH\_FX\_OP\_UNSUPPORTED.

Other handles MAY be opened for ACE4\_READ\_DATA or any other combination of accesses, as long as neither ACE4\_WRITE\_DATA nor ACE4\_APPEND\_DATA are included in the mask.

#### SSH\_FXF\_ACCESS\_BLOCK\_DELETE

The server MUST guarantee that no other handle has been opened with ACE4\_DELETE access, opened with the SSH\_FXF\_ACCESS\_DELETE\_ON\_CLOSE flag set, and that no other handle will be opened with ACE4\_DELETE access or with the SSH\_FXF\_ACCESS\_DELETE\_ON\_CLOSE flag set, and that the file itself is not deleted in any other way until the client closes the handle.

If there is a conflicting lock the server MUST return SSH\_FX\_LOCK\_CONFLICT. If the server cannot make the locking guarantee, it MUST return SSH\_FX\_OP\_UNSUPPORTED.

#### SSH\_FXF\_ACCESS\_BLOCK\_ADVISORY

If this bit is set, the above BLOCK modes are advisory. In advisory mode, only other accesses that specify a BLOCK mode need be considered when determining whether the BLOCK can be granted, and the server need not prevent I/O operations that violate the block mode.

The server MAY perform mandatory locking even if the BLOCK\_ADVISORY bit is set.

#### SSH\_FXF\_ACCESS\_NOFOLLOW

If the final component of the path is a symlink, then the open MUST fail, and the error SSH\_FX\_LINK\_LOOP MUST be returned.

#### SSH\_FXF\_ACCESS\_DELETE\_ON\_CLOSE

The file should be deleted when the last handle to it is closed. (The last handle may not be an sftp-handle.) This MAY be emulated by a server if the OS doesn't support it by deleting the file when this handle is closed.

The 'attrs' field specifies the initial attributes for the file. Default values MUST be supplied by the server for those attributes that are not specified. See Section 'File Attributes' for more information.

The 'attrs' field is ignored if an existing file is opened.



The following table is provided to assist in mapping POSIX semantics to equivalent SFTP file open parameters:

O\_RDONLY

desired-access = READ\_DATA|READ\_ATTRIBUTES

O\_WRONLY

desired-access = WRITE\_DATA|WRITE\_ATTRIBUTES

O\_RDWR

desired-access = READ\_DATA|READ\_ATTRIBUTES|WRITE\_DATA|  
WRITE\_ATTRIBUTES

O\_APPEND

desired-access = WRITE\_DATA|WRITE\_ATTRIBUTES|APPEND\_DATA  
flags = SSH\_FXF\_ACCESS\_APPEND\_DATA and or  
SSH\_FXF\_ACCESS\_APPEND\_DATA\_ATOMIC

O\_CREAT

flags = SSH\_FXF\_OPEN\_OR\_CREATE

O\_TRUNC

flags = SSH\_FXF\_TRUNCATE\_EXISTING

O\_TRUNC|O\_CREATE

flags = SSH\_FXF\_CREATE\_TRUNCATE

### **7.1.2 Opening a Directory**

To enumerate a directory, the client first obtains a handle and then issues directory read requests. When enumeration is complete, the handle MUST be closed.

```
byte    SSH_FXP_OPENDIR
uint32  request-id
string  path [UTF-8]
```

path

The 'path' field is the path name of the directory to be listed (without any trailing slash). See Section 'File Names' for more information on file names.

If 'path' does not refer to a directory, the server MUST return SSH\_FX\_NOT\_A\_DIRECTORY.

The response to this message will be either SSH\_FXP\_HANDLE (if the operation is successful) or SSH\_FXP\_STATUS (if the operation fails).



### [7.1.3](#) Closing Handles

A handle is closed using the following request.

```
byte    SSH_FXP_CLOSE
uint32  request-id
string  handle
```

handle

'handle' is a handle previously returned in the response to SSH\_FXP\_OPEN or SSH\_FXP\_OPENDIR. The handle becomes invalid immediately after this request has been sent.

The response to this request will be a SSH\_FXP\_STATUS message. Note that on some server platforms even a close can fail. For example, if the server operating system caches writes, and an error occurs while flushing cached writes, the close operation may fail.

Note that the handle is invalid regardless of the SSH\_FXP\_STATUS result. There is no way for the client to recover a handle that fails to close. The client **MUST** release all resources associated with the handle regardless of the status. The server **SHOULD** take whatever steps it can to recover from a close failure and to ensure that all resources associated with the handle on the server are correctly released.

## [7.2](#) Reading and Writing

### [7.2.1](#) Reading Files

The following request can be used to read file data:

```
byte    SSH_FXP_READ
uint32  request-id
string  handle
uint64  offset
uint32  length
```

handle

'handle' is an open file handle returned by SSH\_FXP\_OPEN. If 'handle' is not a handle returned by SSH\_FXP\_OPEN, the server **MUST** return SSH\_FX\_INVALID\_HANDLE.



offset

'offset' is the offset (in bytes) relative to the beginning of the file that the read MUST start at. 'offset' is ignored if SSH\_FXF\_TEXT was specified during the open.

length

'length' is the maximum number of bytes to read.

The server MUST not respond with more data than is specified by the 'length' parameter. However, the server MAY respond with less data if EOF is reached, an error is encountered, or the servers internal buffers can not handle such a large request.

If the server specified a non-zero 'max-read-size' in it's 'supported2' ([Section 4.5](#)) extension, then failure to return 'length' bytes indicates that EOF or an error occurred.

### [7.2.2](#) Reading Directories

In order to retrieve a directory listing, the client issues one or more SSH\_FXP\_READDIR requests. In order to obtain a complete directory listing, the client MUST issue repeated SSH\_FXP\_READDIR requests until the server responds with an SSH\_FXP\_STATUS message.

```
byte    SSH_FXP_READDIR
uint32  request-id
string  handle
```

handle

'handle' is a handle returned by SSH\_FXP\_OPENDIR. If 'handle' is an ordinary file handle returned by SSH\_FXP\_OPEN, the server MUST return SSH\_FX\_INVALID\_HANDLE.

The server responds to this request with either a SSH\_FXP\_NAME or a SSH\_FXP\_STATUS message. One or more names may be returned at a time. Full status information is returned for each name in order to speed up typical directory listings.

If there are no more names available to be read, the server MUST respond with a SSH\_FXP\_STATUS message with error code of SSH\_FX\_EOF.

### [7.2.3](#) Writing Files

Writing to a file is achieved using the following message:





```
byte    SSH_FXP_WRITE
uint32  request-id
string  handle
uint64  offset
string  data
```

#### handle

'handle' is an open file handle returned by SSH\_FXP\_OPEN. If 'handle' is not a handle returned by SSH\_FXP\_OPEN, the server MUST return SSH\_FX\_INVALID\_HANDLE.

#### offset

'offset' is the offset (in bytes) relative to the beginning of the file that the write MUST start at. 'offset' is ignored if SSH\_FXF\_TEXT was specified during the open.

The write will extend the file if writing beyond the end of the file. It is legal to write to an offset that extends beyond the end of the file; the semantics are to write the byte value 0x00 from the end of the file to the specified offset and then the data. On most operating systems, such writes do not allocate disk space but instead create a sparse file.

#### data

The data to write to the file.

The server responds to a write request with a SSH\_FXP\_STATUS message.

### [7.3](#) Removing and Renaming Files

The following request can be used to remove a file:

```
byte    SSH_FXP_REMOVE
uint32  request-id
string  filename [UTF-8]
```

#### filename

'filename' is the name of the file to be removed. See Section 'File Names' for more information.

This request cannot be used to remove directories. The server MUST return SSH\_FX\_FILE\_IS\_A\_DIRECTORY in this case.

The server will respond to this request with a SSH\_FXP\_STATUS message.

Files (and directories) can be renamed using the SSH\_FXP\_RENAME



message.

```
byte    SSH_FXP_RENAME
uint32  request-id
string  oldpath [UTF-8]
string  newpath [UTF-8]
uint32  flags
```

where 'request-id' is the request identifier, 'oldpath' is the name of an existing file or directory, and 'newpath' is the new name for the file or directory.

'flags' is 0 or a combination of:

```
SSH_FXP_RENAME_OVERWRITE  0x00000001
SSH_FXP_RENAME_ATOMIC     0x00000002
SSH_FXP_RENAME_NATIVE     0x00000004
```

If flags does not include SSH\_FXP\_RENAME\_OVERWRITE, and there already exists a file with the name specified by newpath, the server MUST respond with SSH\_FX\_FILE\_ALREADY\_EXISTS.

If flags includes SSH\_FXP\_RENAME\_ATOMIC, and the destination file already exists, it is replaced in an atomic fashion. I.e., there is no observable instant in time where the name does not refer to either the old or the new file. SSH\_FXP\_RENAME\_ATOMIC implies SSH\_FXP\_RENAME\_OVERWRITE.

If flags includes SSH\_FXP\_RENAME\_ATOMIC and the server cannot replace the destination in an atomic fashion, then the server MUST respond with SSH\_FX\_OP\_UNSUPPORTED.

Because some servers cannot provide atomic rename, clients should only specify atomic rename if correct operation requires it. If SSH\_FXP\_RENAME\_OVERWRITE is specified, the server MAY perform an atomic rename even if it is not requested.

If flags includes SSH\_FXP\_RENAME\_NATIVE, the server is free to do the rename operation in whatever fashion it deems appropriate. Other flag values are considered hints as to desired behavior, but not requirements.

The server will respond to this request with a SSH\_FXP\_STATUS message.

## **7.4 Creating and Deleting Directories**

New directories can be created using the SSH\_FXP\_MKDIR request. It



has the following format:

```
byte    SSH_FXP_MKDIR
uint32  request-id
string  path [UTF-8]
ATTRS   attrs
```

where 'request-id' is the request identifier.

'path' specifies the directory to be created. See Section 'File Names' for more information on file names.

'attrs' specifies the attributes that should be applied to it upon creation. Attributes are discussed in more detail in Section 'File Attributes'.

The server will respond to this request with a SSH\_FXP\_STATUS message. If a file or directory with the specified path already exists, an error will be returned.

Directories can be removed using the SSH\_FXP\_RMDIR request, which has the following format:

```
byte    SSH_FXP_RMDIR
uint32  request-id
string  path [UTF-8]
```

where 'request-id' is the request identifier, and 'path' specifies the directory to be removed. See Section 'File Names' for more information on file names.

The server responds to this request with a SSH\_FXP\_STATUS message.

## **7.5 Retrieving File Attributes**

Very often, file attributes are automatically returned by SSH\_FXP\_READDIR. However, sometimes there is need to specifically retrieve the attributes for a named file. This can be done using the SSH\_FXP\_STAT, SSH\_FXP\_LSTAT and SSH\_FXP\_FSTAT requests.

SSH\_FXP\_STAT and SSH\_FXP\_LSTAT only differ in that SSH\_FXP\_STAT follows symbolic links on the server, whereas SSH\_FXP\_LSTAT does not follow symbolic links. Both have the same format:



```
byte    SSH_FXP_STAT or SSH_FXP_LSTAT
uint32  request-id
string  path [UTF-8]
uint32  flags
```

where 'request-id' is the request identifier, and 'path' specifies the file system object for which status is to be returned. The server responds to this request with either SSH\_FXP\_ATTRS or SSH\_FXP\_STATUS.

The flags field specify the attribute flags in which the client has particular interest. This is a hint to the server. For example, because retrieving owner / group and acl information can be an expensive operation under some operating systems, the server may choose not to retrieve this information unless the client expresses a specific interest in it.

The client has no guarantee the server will provide all the fields that it has expressed an interest in.

SSH\_FXP\_FSTAT differs from the others in that it returns status information for an open file (identified by the file handle).

```
byte    SSH_FXP_FSTAT
uint32  request-id
string  handle
uint32  flags
```

handle

'handle' is an open file handle from either SSH\_FXP\_OPEN or SSH\_FXP\_OPENDIR.

The server responds to this request with SSH\_FXP\_ATTRS or SSH\_FXP\_STATUS.

## **7.6 Setting File Attributes**

File attributes may be modified using the SSH\_FXP\_SETSTAT and SSH\_FXP\_FSETSTAT requests.

```
byte    SSH_FXP_SETSTAT
uint32  request-id
string  path [UTF-8]
ATTRS   attrs
```





```
byte    SSH_FXP_FSETSTAT
uint32  request-id
string  handle
ATTRS   attrs
```

#### path

The file system object (e.g. file or directory) whose attributes are to be modified. If this object does not exist, or the user does not have sufficient access to write the attributes, the request MUST fail.

#### handle

'handle' is an open file handle from either SSH\_FXP\_OPEN or SSH\_FXP\_OPENDIR. If the handle was not opened with sufficient access to write the requested attributes, the request MUST fail.

#### attrs

Specifies the modified attributes to be applied. Attributes are discussed in more detail in Section 'File Attributes'.

The server will respond with a SSH\_FXP\_STATUS message.

Because some systems must use separate system calls to set various attributes, it is possible that a failure response will be returned, but yet some of the attributes may have been successfully modified. If possible, servers SHOULD avoid this situation; however, clients MUST be aware that this is possible.

## [7.7](#) Dealing with Links

The SSH\_FXP\_READLINK request reads the target of a symbolic link.

```
byte    SSH_FXP_READLINK
uint32  request-id
string  path [UTF-8]
```

where 'request-id' is the request identifier and 'path' specifies the path name of the symlink to be read.

The server will respond with a SSH\_FXP\_NAME packet containing only one name and a dummy attributes value. The name in the returned packet contains the target of the link. If an error occurs, the server MAY respond with SSH\_FXP\_STATUS.

The SSH\_FXP\_LINK request creates a link (either hard or symbolic) on the server.



```
byte    SSH_FXP_LINK
uint32  request-id
string  new-link-path [UTF-8]
string  existing-path [UTF-8]
bool    sym-link
```

#### new-link-path

Specifies the path name of the new link to create.

#### existing-path

Specifies the path of an existing file system object to which the new-link-path will refer.

#### sym-link

Specifies that the link should be a symbolic link, or a special file that redirects file system parsing to the resulting path. It is generally possible to create symbolic links across device boundaries; however, it is not required that a server support this.

If 'sym-link' is false, the link should be a hard link, or a second directory entry referring to the same file or directory object. It is generally not possible to create hard links across devices.

The server shall respond with a SSH\_FXP\_STATUS. Clients should be aware that some servers may return SSH\_FX\_OP\_UNSUPPORTED for either the hard-link, sym-link, or both operations.

The SSH\_FXP\_LOCK creates a byte-range lock on the file specified by the handle. The lock can be either mandatory (meaning that the server enforces that no other process or client can perform operations in violation of the lock) or advisory (meaning that no other process can obtain a conflicting lock, but the server does not enforce that no operation violates the lock).

A server MAY implement an advisory lock in a mandatory fashion; in other words, the server MAY enforce that no operation violates the lock even when operating in advisory mode.

The result is a SSH\_FXP\_STATUS return.

```
byte    SSH_FXP_BLOCK
uint32  request-id
string  handle
uint64  offset
uint64  length
```



uint32 uLockMask

handle

'handle' is an open file handle returned by SSH\_FXP\_OPEN. If 'handle' is not a handle returned by SSH\_FXP\_OPEN, the server MUST return SSH\_FX\_INVALID\_HANDLE.

offset

Beginning of the byte-range to lock.

length

Number of bytes in the range to lock. The special value 0 means lock from 'offset' to the end of the file.

uLockMask

A bit mask of SSH\_FXF\_ACCESS\_BLOCK\_\* values; the meanings are described in [Section 7.1.1.3](#).

The SSH\_FXP\_UNLOCK removes a previously acquired byte-range lock on the specified handle.

The result is a SSH\_FXP\_STATUS return.

byte SSH\_FXP\_UNBLOCK

uint32 request-id

string handle

uint64 offset

uint64 length

handle

'handle' is an open file handle returned by SSH\_FXP\_OPEN. If 'handle' is not a handle returned by SSH\_FXP\_OPEN, the server MUST return SSH\_FX\_INVALID\_HANDLE.

offset

Beginning of the byte-range to lock.

length

Number of bytes in the range to lock. The special value 0 means lock from 'offset' to the end of the file.

## [7.8](#) Canonicalizing the Server-Side Path Name

The SSH\_FXP\_REALPATH request can be used to have the server canonicalize any given path name to an absolute path. This is useful for converting path names containing "." components or relative pathnames without a leading slash into absolute paths. The format of the request is as follows:



```
byte    SSH_FXP_REALPATH
uint32  request-id
string  original-path [UTF-8]
string  compose-path [optional]
byte    control-byte [optional]
```

#### original-path

The first component of the path which the client wishes resolved into a absolute canonical path. This may be the entire path.

#### compose-path

A path which the client wishes the server to compose with the original path to form the new path. This field is optional, and if it is not present in the packet, it is assumed to be a zero length string.

#### control-byte

```
SSH_FXP_REALPATH_NO_CHECK    0x00000001
SSH_FXP_REALPATH_STAT_IF    0x00000002
SSH_FXP_REALPATH_STAT_ALWAYS 0x00000003
```

This field is optional, and if it is not present in the packet, it is assumed to be SSH\_FXP\_REALPATH\_NO\_CHECK.

If SSH\_FXP\_REALPATH\_NO\_CHECK is specified, the server MUST NOT fail the request if the path does not exist, is hidden, or the user does not have access to the path or some component thereof. However, the path MAY NOT be completely resolved to it's component form. For example, symlinks may not be followed in this case. The server MAY fail the request if the path is not syntactically valid, or for other reasons.

If SSH\_FXP\_REALPATH\_STAT\_IF is specified, the server MUST stat the path if it exists and is accessible to the client. However, if the path does not exist, isn't visible, or isn't accessible, the server MUST NOT fail the request. If the stat failed, the file type will be SSH\_FILEXFER\_TYPE\_UNKNOWN. If the client needs to distinguish between files that are actually SSH\_FILEXFER\_TYPE\_UNKNOWN and paths that don't exist, it will have to issue a separate stat command in this case.

If SSH\_FXP\_REALPATH\_STAT\_ALWAYS is specified the server MUST stat the path. If the stat operation fails, the server MUST fail the request.

The server MUST take the 'original-path' and apply the 'compose-path'





as a modification to it. 'compose-path' MAY be relative to 'original-path' or may be an absolute path, in which case 'original-path' will be discarded. The 'compose-path' may be zero length.

The server will respond with a SSH\_FXP\_NAME packet containing the canonical form of the composite path. If SSH\_FXP\_REALPATH\_NO\_CHECK is specified, the attributes are dummy values.

### **7.8.1 Best Practice for Dealing with Paths**

BEGIN: RFCEDITOR REMOVE BEFORE PUBLISHING

Previous to this version, clients typically composed new paths themselves and then called both realpath and stat on the resulting path to get its canonical name and see if it really existed and was a directory.

This required clients to assume certain things about how a relative vs. realpath looked. The new realpath allows clients to no longer make those assumptions and to remove one round trip from the process and get deterministic behavior from all servers.

END: RFCEDITOR REMOVE BEFORE PUBLISHING

The client SHOULD treat the results of SSH\_FXP\_REALPATH as a canonical absolute path, even if the path does not appear to be absolute. A client that uses REALPATH(".", "") and treats the result as absolute, even if there is no leading slash, will continue to function correctly, even when talking to a Windows NT or VMS style system, where absolute paths may not begin with a slash.

The client SHOULD also use SSH\_FXP\_REALPATH call to compose paths so that it does not need to know when a path is absolute or relative.

For example, if the client wishes to change directory up, and the server has returned "c:/x/y/z" from REALPATH, the client SHOULD use REALPATH("c:/x/y/z", "..", SSH\_FXP\_REALPATH\_STAT\_ALWAYS)

As a second example, if the client wishes transfer local file "a" to remote file "/b/d/e", and server has returned "dka100:/x/y/z" as the canonical path of the current directory, the client SHOULD send REALPATH("dka100:/x/y/z", "/b/d/e", SSH\_FXP\_REALPATH\_STAT\_IF). This call will determine the correct path to use for the open request and whether the /b/d/e represents a directory.

## **8. Responses from the Server to the Client**

The server responds to the client using one of a few response



packets. All requests can return a SSH\_FXP\_STATUS response upon failure. When the operation is successful, and no data needs to be returned, the SSH\_FXP\_STATUS response with SSH\_FX\_OK status is appropriate.

Exactly one response will be returned for each request. Each response packet contains a request identifier which can be used to match each response with the corresponding request. Note that it is legal to have several requests outstanding simultaneously, and the server is allowed to send responses to them in a different order from the order in which the requests were sent (the result of their execution, however, is guaranteed to be as if they had been processed one at a time in the order in which the requests were sent).

Response packets are of the same general format as request packets. Each response packet begins with the request identifier.

### **8.1 Status Response**

The format of the data portion of the SSH\_FXP\_STATUS response is as follows:

```
byte    SSH_FXP_STATUS
uint32  request-id
uint32  error/status code
string  error message (ISO-10646 UTF-8 [RFC-2279])
string  language tag (as defined in [RFC-1766])
        error-specific data
```

#### **request-id**

The 'request-id' specified by the client in the request the server is responding to.

#### **error/status code**

Machine readable status code indicating the result of the request. Error code values are defined below. The value SSH\_FX\_OK indicates success, and all other values indicate failure.

#### **error message**

Human readable description of the error.

#### **language tag**

'language tag' specifies the language the error is in.

#### **<error-specific data>**

The error-specific data may be empty, or may contain additional information about the error. For error codes that send error-specific data, the format of the data is defined below.



## Error codes:

SSH_FX_OK	0
SSH_FX_EOF	1
SSH_FX_NO_SUCH_FILE	2
SSH_FX_PERMISSION_DENIED	3
SSH_FX_FAILURE	4
SSH_FX_BAD_MESSAGE	5
SSH_FX_NO_CONNECTION	6
SSH_FX_CONNECTION_LOST	7
SSH_FX_OP_UNSUPPORTED	8
SSH_FX_INVALID_HANDLE	9
SSH_FX_NO_SUCH_PATH	10
SSH_FX_FILE_ALREADY_EXISTS	11
SSH_FX_WRITE_PROTECT	12
SSH_FX_NO_MEDIA	13
SSH_FX_NO_SPACE_ON_FILESYSTEM	14
SSH_FX_QUOTA_EXCEEDED	15
SSH_FX_UNKNOWN_PRINCIPAL	16
SSH_FX_LOCK_CONFLICT	17
SSH_FX_DIR_NOT_EMPTY	18
SSH_FX_NOT_A_DIRECTORY	19
SSH_FX_INVALID_FILENAME	20
SSH_FX_LINK_LOOP	21
SSH_FX_CANNOT_DELETE	22
SSH_FX_INVALID_PARAMETER	23
SSH_FX_FILE_IS_A_DIRECTORY	24
SSH_FX_BYTE_RANGE_LOCK_CONFLICT	25
SSH_FX_BYTE_RANGE_LOCK_REFUSED	26
SSH_FX_DELETE_PENDING	27
SSH_FX_FILE_CORRUPT	28

## SSH\_FX\_OK

Indicates successful completion of the operation.

## SSH\_FX\_EOF

An attempt to read past the end-of-file was made; or, there are no more directory entries to return.

## SSH\_FX\_NO\_SUCH\_FILE

A reference was made to a file which does not exist.

## SSH\_FX\_PERMISSION\_DENIED

The user does not have sufficient permissions to perform the operation.



**SSH\_FX\_FAILURE**

An error occurred, but no specific error code exists to describe the failure.

This error message SHOULD always have meaningful text in the the 'error message' field.

**SSH\_FX\_BAD\_MESSAGE**

A badly formatted packet or other SFTP protocol incompatibility was detected.

**SSH\_FX\_NO\_CONNECTION**

There is no connection to the server. This error MAY be used locally, but MUST NOT be return by a server.

**SSH\_FX\_CONNECTION\_LOST**

The connection to the server was lost. This error MAY be used locally, but MUST NOT be return by a server.

**SSH\_FX\_OP\_UNSUPPORTED**

An attempted operation could not be completed by the server because the server does not support the operation.

This error MAY be generated locally by the client if e.g. the version number exchange indicates that a required feature is not supported by the server, or it may be returned by the server if the server does not implement an operation).

**SSH\_FX\_INVALID\_HANDLE**

The handle value was invalid.

**SSH\_FX\_NO\_SUCH\_PATH**

The file path does not exist or is invalid.

**SSH\_FX\_FILE\_ALREADY\_EXISTS**

The file already exists.

**SSH\_FX\_WRITE\_PROTECT**

The file is on read-only media, or the media is write protected.

**SSH\_FX\_NO\_MEDIA**

The requested operation cannot be completed because there is no media available in the drive.





**SSH\_FX\_NO\_SPACE\_ON\_FILESYSTEM**

The requested operation cannot be completed because there is no free space on the filesystem.

**SSH\_FX\_QUOTA\_EXCEEDED**

The operation cannot be completed because it would exceed the user's storage quota.

**SSH\_FX\_UNKNOWN\_PRINCIPAL**

A principal referenced by the request (either the 'owner', 'group', or 'who' field of an ACL), was unknown. The error specific data contains the problematic names. The format is one or more:

string unknown-name

Each string contains the name of a principle that was unknown.

**SSH\_FX\_LOCK\_CONFLICT**

The file could not be opened because it is locked by another process.

**SSH\_FX\_DIR\_NOT\_EMPTY**

The directory is not empty.

**SSH\_FX\_NOT\_A\_DIRECTORY**

The specified file is not a directory.

**SSH\_FX\_INVALID\_FILENAME**

The filename is not valid.

**SSH\_FX\_LINK\_LOOP**

Too many symbolic links encountered.

**SSH\_FX\_CANNOT\_DELETE**

The file cannot be deleted. One possible reason is that the advisory READONLY attribute-bit is set.

**SSH\_FX\_INVALID\_PARAMETER**

One of the parameters was out of range, or the parameters specified cannot be used together.

**SSH\_FX\_FILE\_IS\_A\_DIRECTORY**

The specified file was a directory in a context where a directory cannot be used.



**SSH\_FX\_BYTE\_RANGE\_LOCK\_CONFLICT**

Some operating systems support locking a range of bytes within a file. On such operating systems, it is possible for a SFTP request to fail due to some other process owning a byte-range lock, even though the SFTP protocol does not support byte-range locks natively.

A read or write operation failed because another process's byte-range lock overlaps with the request.

**SSH\_FX\_BYTE\_RANGE\_LOCK\_REFUSED**

A request for a byte range lock was refused.

**SSH\_FX\_DELETE\_PENDING**

An operation was attempted on a file for which a delete operation is pending.

**SSH\_FX\_FILE\_CORRUPT**

The file is corrupt; an filesystem integrity check should be run.

## **8.2 Handle Response**

The SSH\_FXP\_HANDLE response has the following format:

```
byte    SSH_FXP_HANDLE
uint32  request-id
string  handle
```

'handle'

An arbitrary string that identifies an open file or directory on the server. The handle is opaque to the client; the client MUST NOT attempt to interpret or modify it in any way. The length of the handle string MUST NOT exceed 256 data bytes.

## **8.3 Data Response**

The SSH\_FXP\_DATA response has the following format:

```
byte    SSH_FXP_DATA
uint32  request-id
string  data
bool    end-of-file [optional]
```



**data**

'data' is an arbitrary byte string containing the requested data. The data string may be at most the number of bytes requested in a SSH\_FXP\_READ request, but may also be shorter. (See [Section 7.2.1.](#))

**end-of-file**

This field is optional. If it is present in the packet, it MUST be true, and it indicates that EOF was reached during this read. This can help the client avoid a round trip to determine whether a short read was normal (due to EOF) or some other problem (limited server buffer for example.)

## **8.4 Name Response**

The SSH\_FXP\_NAME response has the following format:

```
byte    SSH_FXP_NAME
uint32  request-id
uint32  count
repeats count times:
    string    filename [UTF-8]
    ATTRS     attrs
    bool end-of-list [optional]
```

**count**

The number of names returned in this response, and the 'filename' and 'attrs' field repeat 'count' times.

**filename**

A file name being returned (for SSH\_FXP\_READDIR, it will be a relative name within the directory, without any path components; for SSH\_FXP\_REALPATH it will be an absolute path name.)

**attrs**

The attributes of the file as described in Section 'File Attributes'.

**end-of-list**

This field is optional. If present in the packet, it MUST be true, and indicates that there are no more entries to be read. This can save the client a round trip to determine if there are more entries to be read.



### 8.5 Attrs Response

The SSH\_FXP\_ATTRS response has the following format:

```
byte    SSH_FXP_ATTRS
uint32  request-id
ATTRS   attrs
```

attrs

The returned file attributes as described in Section 'File Attributes'.

## 9. Extensions

The SSH\_FXP\_EXTENDED request provides a generic extension mechanism for adding additional commands.

```
byte    SSH_FXP_EXTENDED
uint32  request-id
string  extended-request
... any request-specific data ...
```

request-id

Identifier to be returned from the server with the response.

extended-request

A string naming the extension. Vendor-specific extensions have use the "name@domain" syntax, where domain is an internet domain name of the vendor defining the request.

The IETF may also define extensions to the protocol. These extension names will not have an '@' in them.

request-specific data

The rest of the request is defined by the extension; servers SHOULD NOT attempt to interpret it if they do not recognize the 'extended-request' name.

The server may respond to such requests using any of the response packets defined in Section 'Responses from the Server to the Client'. Additionally, the server may also respond with a SSH\_FXP\_EXTENDED\_REPLY packet, as defined below. If the server does not recognize the 'extended-request' name, then the server MUST respond with SSH\_FXP\_STATUS with error/status set to SSH\_FX\_OP\_UNSUPPORTED.

The SSH\_FXP\_EXTENDED\_REPLY packet can be used to carry arbitrary





extension-specific data from the server to the client. It is of the following format:

```
byte    SSH_FXP_EXTENDED_REPLY
uint32  request-id
... any request-specific data ...
```

There is a range of packet types reserved for use by extensions. In order to avoid collision, extensions that use additional packet types should determine those numbers dynamically.

The suggested way of doing this is have an extension request from the client to the server that enables the extension; the extension response from the server to the client would specify the actual type values to use, in addition to any other data.

Extension authors should be mindful of the limited range of packet types available (there are only 45 values available) and avoid requiring a new packet type where possible.

## **9.1 File Hashing**

BEGIN: RFCEDITOR REMOVE BEFORE PUBLISHING

After some discussion of this at connectathon, I know of two uses for this feature, neither one of which the feature is entirely suited for:

- o Checking that a file has been uploaded to the server correctly; some portion of the customers wanting this feature want it in a security sense, as part of proof the server has the file.
- o Optimizing upload or download of the file; multiple hashes are performed on small pieces of the file and the results are used to determine what chunks of the file, if any, need to be transferred. This is similar to the way rsync works.

I've seen both of these implemented.

For the first case, the extension has several drawbacks, including:

- o A FIPS implementation can't ship md5.
- o MD5's security is potential weaker than other options.
- o Being hard-coded to MD5 makes it impossible to adapt to future developments in the arena of MD5 compromises.

For the second case, the extension has these drawbacks:

- o MD5 is expensive (relative to other options.)
- o The extension must be sent potentially thousands of times to retrieve the desired granularity of hashes.



Therefore, for this draft, this section is marked experimental; I've included a second proposed extension. Please post your thoughts on the mailing list. (I did it this way just so I could get a draft out that I and my active co-author are happy with.

In addition, implementation experience has shown the quick check hash to not be useful.

END: RFCEDITOR REMOVE BEFORE PUBLISHING

#### **9.1.1 Checking File Contents: v5 extension**

This extension allows a client to easily check if a file (or portion thereof) that it already has matches what is on the server.

```
byte    SSH_FXP_EXTENDED
uint32  request-id
string  "md5-hash" / "md5-hash-handle"
string  filename [UTF-8] / file-handle
uint64  start-offset
uint64  length
string  quick-check-hash
```

##### **filename**

Used if "md5-hash" is specified; indicates the name of the file to use. The hash will be of the file contents as it would appear on the wire if the file were opened with no special flags.

##### **file-handle**

Used if "md5-hash-handle" is specified; specifies a file handle to read the data from. The handle MUST be a file handle, and ACE4\_READ\_DATA MUST have been included in the desired-access when the file was opened.

If this file handle was opened in TEXT mode, the md5-hash must be made of the data as it would be sent on the wire.

##### **start-offset**

The starting offset of the data to hash.

##### **length**

The length of data to include in the hash. If both start-offset and length are zero, the entire file should be included.



#### quick-check-hash

The hash over the first 2048 bytes of the data range as the client knows it, or the entire range, if it is less than 2048 bytes.

This allows the server to quickly check if it is worth the resources to hash a big file.

If this is a zero length string, the client does not have the data, and is requesting the hash for reasons other than comparing with a local file. The server MAY return SSH\_FX\_OP\_UNSUPPORTED in this case.

The response is either a SSH\_FXP\_STATUS packet, indicating an error, or the following extended reply packet:

```
byte    SSH_FXP_EXTENDED_REPLY
uint32  request-id
string  "md5-hash"
string  hash
```

If 'hash' is zero length, then the 'quick-check-hash' did not match, and no hash operation was performed. Otherwise, 'hash' contains the hash of the entire data range (including the first 2048 bytes that were included in the 'quick-check-hash'.)

#### [9.1.2](#) Checking File Contents

This extension allows a client to easily check if a file (or portion thereof) that it already has matches what is on the server.

```
byte    SSH_FXP_EXTENDED
uint32  request-id
string  "check-file"
string  handle
string  hash-algorithm-list
uint64  start-offset
uint64  length
uint32  block-size
```

#### handle

'handle' is an open file handle returned by SSH\_FXP\_OPEN. If 'handle' is not a handle returned by SSH\_FXP\_OPEN, the server MUST return SSH\_FX\_INVALID\_HANDLE. If ACE4\_READ\_DATA MUST was not included when the file was opened, the server MUST return STATUS\_PERMISSION\_DENIED.



If this file handle was opened in TEXT mode, the check must be performed on the data as it would be sent on the wire.

#### hash-algorithm-list

A comma separated list of hash algorithms the client is willing to accept for this operation. The server MUST pick the first hash on the list that it supports.

Currently supported algorithms are "md5", "sha1", "sha224", "sha256", "sha384", "sha512", and "crc32". Additional algorithms may be added by following the DNS extensibility naming convention outlined in [[I-D.ietf-secsh-architecture](#)].

MD5 is described in [[RFC1321](#)]. SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 are described in [[FIPS-180-2](#)]. crc32 is described in [[ISO.3309.1991](#)], and is the same algorithm used in [[RFC1510](#)]

#### start-offset

The starting offset of the data to include in the hash.

#### length

The length of data to include in the hash. If length is zero, all the data from start-offset to the end-of-file should be included.

#### block-size

An independent hash MUST be computed over every block in the file. The size of blocks is specified by block-size. The block-size MUST NOT be smaller than 256 bytes. If the block-size is 0, then only one hash, over the entire range, MUST be made.

The response is either a SSH\_FXP\_STATUS packet, indicating an error, or the following extended reply packet:

```
byte    SSH_FXP_EXTENDED_REPLY
uint32  request-id
string  "check-file"
string  hash-algo-used
byte    hash[n][block-count]
```

#### hash-algo-used

The hash algorithm that was actually used.

#### hash

The computed hashes. The hash algorithm used determines the size of n. The number of block-size chunks of data in the file determines block-count. The hashes are placed in the packet one after another, with no decoration.





Note that if the length of the range is not an even multiple of block-size, the last hash will have been computed over only the remainder of the range instead of a full block.

## 9.2 Querying Available Space

The following extension provides a way to discover the available space for an arbitrary path.

```
byte    SSH_FXP_EXTENDED
uint32  request-id
string  "space-available"
string  path      [UTF-8]
```

path

'path' for which the available space should be reported. This 'path' is not required to be the mount point path, but MAY be a directory or file contained within the mount.

The reply to the request is as follows:

```
byte    SSH_FXP_EXTENDED_REPLY
uint32  request-id
uint64  bytes-on-device
uint64  unused-bytes-on-device
uint64  bytes-available-to-user
uint64  unused-bytes-available-to-user
uint32  bytes-per-allocation-unit
```

bytes-on-device

The total number of bytes on the device which stores 'path', both used and unused, or 0 if unknown.

unused-bytes-on-device

The total number of unused bytes available on the device which stores 'path', or 0 if unknown.

bytes-available-to-user

The total number of bytes, both used and unused, available to the authenticated user on the device which stores 'path', or 0 if unknown.

unused-bytes-available-to-user

The total number of unused bytes available to the authenticated user on the device which stores 'path', or 0 if unknown.



**bytes-per-allocation-unit**

The number of bytes in each allocation unit on the device, or in other words, the minimum number of bytes that a file allocation size can grow or shrink by. If the server does not know this information, or the file-system in use does not use allocation blocks, this value MUST be 0.

**9.3 Querying User Home Directory**

Many users are used to being able to type '~' as an alias for their home directory, or ~username as an alias for another user's home directory. To support this feature, a server MAY support following extension.

```
byte    SSH_FXP_EXTENDED
uint32  request-id
string  "home-directory"
string  username      [UTF-8]
```

**username**

Username whose home directory path is being requested. An empty string implies the current user.

The reply to the request is either a SSH\_FXP\_STATUS packet or the following extended reply:

```
byte    SSH_FXP_EXTENDED_REPLY
uint32  request-id
string  "home-directory"
string  absolute-pathname
```

**absolute-pathname**

Absolute pathname of the specified user's home directory, suitable for use in operations such as REALPATH or OPEN.

**10. Implementation Considerations**

In order for this protocol to perform well, especially over high latency networks, multiple read and write requests should be queued to the server.

The data size of requests should match the maximum packet size for the next layer up in the protocol chain.

When implemented over ssh, the best performance should be achieved when the data size matches the channel's max packet, and the channel



window is a multiple of the channel packet size.

Implementations MUST be aware that requests do not have to be satisfied in the order issued. (See Request Synchronization and Reordering ([Section 3.1](#)).)

Implementations MUST also be aware that read requests may not return all the requested data, even if the data is available.

## **11. Security Considerations**

It is assumed that both ends of the connection have been authenticated and that the connection has privacy and integrity features. Such security issues are left to the underlying transport protocol, except to note that if this is not the case, an attacker could manipulate files on the server at will and thus wholly compromise the server.

This protocol provides file system access to arbitrary files on the server (constrained only by the server implementation). It is the responsibility of the server implementation to enforce any access controls that may be required to limit the access allowed for any particular user (the user being authenticated externally to this protocol, typically using [[I-D.ietf-secsh-userauth](#)]).

Extreme care must be used when interpreting file handle strings. In particular, care must be taken that a file handle string is valid in the context of a given 'file-share' session. For example, the 'file-share' server daemon may have files which it has opened for its own purposes, and the client must not be able to access these files by specifying an arbitrary file handle string.

The permission field of the attrib structure ([Section 6.6](#)) may include the SUID, SGID, and SVTX (sticky) bits. Clients should use extreme caution when setting these bits on either remote or local files. (I.e., just because a file was SUID on the remote system does not necessarily imply that it should be SUID on the local system.)

Filesystems often contain entries for objects that are not files at all, but are rather devices. For example, it may be possible to access serial ports, tape devices, or named pipes using this protocol. Servers should exercise caution when granting access to such resources. In addition to the dangers inherent in allowing access to such a device, some devices may be 'slow', and could cause denial of service by causing the server to block for a long period of time while I/O is performed to such a device.

Servers should take care that file-system quotas are respected for



users. In addition, implementations should be aware that attacks may be possible, or facilitated, by filling a filesystem. For example, filling the filesystem where event logging and auditing occurs may, at best, cause the system to crash, or at worst, allow the attacker to take untraceable actions in the future.

Servers should take care that filenames are in their appropriate canonical form, and to ensure that filenames not in canonical form cannot be used to bypass access checks or controls.

If the server implementation limits access to certain parts of the file system, extra care must be taken in parsing file names which contain the `'..'` path element, and when following symbolic links, shortcuts, or other filesystem objects which might transpose the path to refer to an object outside of the restricted area. There have been numerous reported security bugs where a `".."` in a path name has allowed access outside the intended area.

## **12. Changes from Previous Protocol Versions**

The SSH File Transfer Protocol has changed over time, before its standardization. The following is a description of the incompatible changes between different versions.

### **12.1 Changes Between Versions 6 and 5**

- o **Add ability to negotiate version when client supports discontinuous ranges of protocol version.**
- o Add `'filename-charset'` and the `'filename-translation-control'` extensions to allow better support of servers that can't reliably translate to UTF-8.
- o Add `DIR_NOT_EMPTY`, `NOT_A_DIRECTORY`, `INVALID_FILENAME`, `LINK_LOOP`, `CANNOT_DELETE`, `INVALID_PARAMETER`, `FILE_IS_A_DIRECTORY`, `BYTE_RANGE_LOCK_CONFLICT`, `BYTE_RANGE_LOCK_REFUSED`, `DELETE_PENDING`, and `FILE_CORRUPT` error codes.
- o Added space-available extension.
- o Added `NOFOLLOW` and `DELETE_ON_CLOSE` flag to open flags.
- o Added allocation-size, text-hint, link-count, mime-type, and untranslated-name fields to `attrib` structure. Add `ATTR_FLAGS_TRANSLATION_ERR` to the `attrib-bits`.
- o Add optional `'compose-path'` and `'control-byte'` to `REALPATH`; make `realpath`'s behaviour truly deterministic (i.e., `MUST` instead of `SHOULD`.) Give clients the ability to compose path's without understanding what is relative and what is absolute.
- o Give `SSH_FXP_DATA` and `SSH_FXP_NAME` optional end-of-data-set flags, which can help the client avoid a round trip during normal operation.





- o Changed the SYMLINK packet to be LINK and give it the ability to create hard links. Also change it's packet number because many implementation implemented SYMLINK with the arguments reversed. Hopefully the new argument names make it clear which way is which.
- o Clarify who should apply umask to POSIX mode bits (the client, not the server.)
- o Specify behavior for otherwise valid packets with excess data and unrecognized packet types.
- o Add home directory extension.
- o Remove "#define" from symbol definitions to shorten line and help us pass idnits.
- o Change "supported" extension to "supported2", remove desired-access-bits, separate attrib extension from SSH\_FXP\_EXTENDED extensions.
- o Add "vendor-id" extension.
- o Add ctime and attrib-bits-valid to attrib structure.
- o Unrecognized attrib extensions cause failure rather than ignored.
- o Option 'end of data' flags on data and names responses.

## **12.2 Changes Between Versions 5 and 4**

Many of the changes between version 5 and version 4 are to better support the changes in version 4, and to better specify error conditions.

- o Add "supported" extension to communicate features supported.
- o Clarify error handling when client requests unsupported feature. (For example, attempts to write an unsupported attribute.)
- o Add attrib-bits field to the attribute structure, which specifies a number of boolean attributes related to files and directories, including advisory read-only and case-sensitivity bits.
- o Clarify the actual bit values to be used for the permissions field (since POSIX doesn't define values) and correct the value of ATTR\_PERMISSIONS flag.
- o Some reordering of sections to attempt to get a better grouping of related functionality.
- o Open request explicitly specifies the access desired for the file.
- o Add support for explicitly requesting file locking.
- o Add support for better control of the rename operation.
- o Add SSH\_FX\_NO\_SPACE\_ON\_FILESYSTEM, SSH\_FX\_QUOTA\_EXCEEDED, and SSH\_FX\_UNKNOWN\_PRINCIPLE error codes.
- o Add support for error specific data. This is used by a new SSH\_FX\_UNKNOWN\_PRINCIPLE error to communicate which principles are unknown.
- o Add support for retrieving md5-hash of file contents.
- o Update security section.



### **12.3 Changes Between Versions 4 and 3**

Many of the changes between version 4 and version 3 are to the attribute structure to make it more flexible for non-unix platforms.

- o Clarify the use of stderr by the server.
- o Clarify handling of very large read requests by the server.
- o Make all filenames UTF-8.
- o Added 'newline' extension.
- o Made time fields 64 bit, and optionally have nanosecond resolution.
- o Made file attribute owner and group strings so they can actually be used on disparate systems.
- o Added createtime field, and added separate flags for atime, createtime, and mtime so they can be set separately.
- o Split the file type out of the permissions field and into its own field (which is always present.)
- o Added acl attribute.
- o Added SSH\_FXF\_TEXT file open flag.
- o Added flags field to the get stat commands so that the client can specifically request information the server might not normally include for performance reasons.
- o Removed the long filename from the names structure-- it can now be built from information available in the attrs structure.
- o Added reserved range of packet numbers for extensions.
- o Added several additional error codes.

### **12.4 Changes Between Versions 3 and 2**

- o The SSH\_FXP\_READLINK and SSH\_FXP\_SYMLINK messages were added.
- o The SSH\_FXP\_EXTENDED and SSH\_FXP\_EXTENDED\_REPLY messages were added.
- o The SSH\_FXP\_STATUS message was changed to include fields 'error message' and 'language tag'.

### **12.5 Changes Between Versions 2 and 1**

- o The SSH\_FXP\_RENAME message was added.

### **12.6 Changes Between Versions 1 and 0**

- o Implementation changes, no actual protocol changes.

## **13. References**



### **13.1 Normative References**

- [RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", [RFC 1321](#), April 1992.
- [RFC1510] Kohl, J. and B. Neuman, "The Kerberos Network Authentication Service (V5)", [RFC 1510](#), September 1993.
- [RFC3010] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "NFS version 4 Protocol", [RFC 3010](#), December 2000.
- [I-D.ietf-secsh-architecture]  
Lonvick, C., "SSH Protocol Architecture",  
[draft-ietf-secsh-architecture-22](#) (work in progress),  
March 2005.
- [I-D.ietf-secsh-transport]  
Lonvick, C., "SSH Transport Layer Protocol",  
[draft-ietf-secsh-transport-24](#) (work in progress),  
March 2005.
- [I-D.ietf-secsh-connect]  
Lonvick, C., "SSH Connection Protocol",  
[draft-ietf-secsh-connect-25](#) (work in progress),  
March 2005.
- [IEEE.1003-1.1996]  
Institute of Electrical and Electronics Engineers,  
"Information Technology - Portable Operating System  
Interface (POSIX) - Part 1: System Application Program  
Interface (API) [C Language]", IEEE Standard 1003.2, 1996.
- [FIPS-180-2]  
National Institute of Standards and Technology, "Secure  
Hash Standard (SHS)", Federal Information Processing  
Standards Publication 180-2, August 2002.
- [ISO.3309.1991]  
International Organization for Standardization,  
"Information Technology - Telecommunications and  
information exchange between systems - High-level data  
link control (HDLC) procedures - Frame structure",  
ISO Standard 3309, June 1991.



### **13.2 Informative References**

- [RFC1521] Borenstein, N. and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", [RFC 1521](#), September 1993.
- [RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", [RFC 2246](#), January 1999.
- [RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", [BCP 18](#), [RFC 2277](#), January 1998.
- [I-D.ietf-secsh-userauth]  
Lonvick, C., "SSH Authentication Protocol",  
[draft-ietf-secsh-userauth-27](#) (work in progress),  
March 2005.

#### Authors' Addresses

Joseph Galbraith  
VanDyke Software  
4848 Tramway Ridge Blvd  
Suite 101  
Albuquerque, NM 87111  
US

Phone: +1 505 332 5700  
Email: galb-list@vandyke.com

Oskari Saarenmaa  
F-Secure  
Tammasaarenkatu 7  
Helsinki 00180  
FI

Email: oskari.saarenmaa@f-secure.com





Tatu Ylonen  
SSH Communications Security Corp  
Fredrikinkatu 42  
HELSINKI FIN-00100  
Finland

Email: ylo@ssh.com

Sami Lehtinen  
SSH Communications Security Corp  
Fredrikinkatu 42  
HELSINKI FIN-00100  
Finland

Email: sjl@ssh.com

Trademark notice

"ssh" is a registered trademark in the United States and/or other countries.



## Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at [ietf-ipr@ietf.org](mailto:ietf-ipr@ietf.org).

## Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

## Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

