

Secure Shell Working Group
Internet-Draft
Expires: January 11, 2007

J. Galbraith
VanDyke Software
O. Saarenmaa
F-Secure
July 10, 2006

SSH File Transfer Protocol
draft-ietf-secsh-filexfer-13.txt

Status of this Memo

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 becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

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/lid-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 January 11, 2007.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

The SSH File Transfer Protocol provides secure file transfer functionality over any reliable, bidirectional octet 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

1. Introduction	4
2. Acknowledgements	4
3. Use with the SSH Connection Protocol	4
3.1. The Use of 'stderr' in the server	5
4. General Packet Format	5
4.1. Request Synchronization and Reordering	6
4.2. New data types defined by this document	7
4.3. Packet Types	7
5. Protocol Initialization	9
5.1. Client Initialization	9
5.2. Server Initialization	9
5.3. Determining Server Newline Convention	10
5.4. Supported Features	10
5.5. Version re-negotiation	14
6. File Names	15
7. File Attributes	16
7.1. valid-attribute-flags	17
7.2. Type	18
7.3. Size	18
7.4. allocation-size	19
7.5. Owner and Group	19
7.6. Permissions	20
7.7. Times	21
7.8. ACL	21
7.9. attrib-bits and attrib-bits-valid	24
7.10. text-hint	27
7.11. mime-type	27
7.12. link-count	27
7.13. untranslated-name	28
7.14. Extended Attributes	28
8. Requests From the Client to the Server	28
8.1. Opening and Closing Files and Directories	28
8.1.1. Opening a File	29
8.1.2. Opening a Directory	35
8.1.3. Closing Handles	35
8.2. Reading and Writing	36
8.2.1. Reading Files	36
8.2.2. Reading Directories	37
8.2.3. Writing Files	37
8.3. Removing and Renaming Files	38
8.4. Creating and Deleting Directories	39
8.5. Retrieving File Attributes	40
8.6. Setting File Attributes	41
8.7. Dealing with Links	42
8.8. Byte-range locks	43
8.8.1. Obtaining a byte range lock	43

8.8.2. Releasing a byte range lock	44
8.9. Canonicalizing the Server-Side Path Name	45
8.9.1. Best Practice for Dealing with Paths	46
9. Responses from the Server to the Client	47
9.1. Status Response	48
9.2. Handle Response	52
9.3. Data Response	52
9.4. Name Response	53
9.5. Attrs Response	53
10. Extensions	54
11. Implementation Considerations	55
12. IANA Considerations	55
13. Security Considerations	55
14. Changes from Previous Protocol Versions	57
15. References	57
15.1. Normative References	57
15.2. Informative References	57
Authors' Addresses	59
Intellectual Property and Copyright Statements	60

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 [RFC4251], 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 [RFC4251].

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. Acknowledgements

This document owes its initial creation and protocol design to Tatu Ylonen and Sami Lehtinen of SSH Communications Security Corp.

We express our gratitude to them for their initial work on this protocol.

3. 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 [RFC4254] in the section "Starting a Shell or a Command". The subsystem name used with this protocol is "sftp".

3.1. The Use of 'stderr' in the server

This protocol uses stdout and stdin to transmit binary protocol data. The "session" channel ([RFC4254]), 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.

4. 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 [RFC4253].)

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.

Additionally, when a packet has two or more optional fields, and an implementation wishes to use the *i*-th optional field, all fields from 1 to *i* MUST be present. In other words, only fields after the last field the implementation wishes to send are actually optional.

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.

4.1. 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.

4.2. New data types defined by this document

This document defines these data types in addition to those defined in [RFC4251].

uint16

Represents a 16-bit unsigned integer. Stored as 2 bytes in the order of decreasing significance (network byte order).

int64

Represents a 64-bit signed integer. Stored using two's complement, 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 [RFC4251].

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

4.3. 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 use of additional packet types in the non-extension range MUST be introduced through IETF consensus. New packet types to be sent from the client to the server MAY be introduced without changing the

protocol version (Section 5). Because the client has no way to respond to unrecognized packet types, new packet types to be sent from the server to the client the client **MUST** not use unless the protocol version is changed or the client has negotiated to receive them. This negotiation **MAY** be explicit, through the use of extensions, or **MAY** be implicit, by the client itself using a packet type not defined above.

5. 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.

5.1. 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 noncontiguous 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.

5.2. 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 4.2). Implementations MUST silently ignore any extensions whose names they do not recognize.

5.3. 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_FXF_TEXT_MODE file open flag (Section 8.1.1) 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.

5.4. 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 it as part of their version packet.

```
string "supported2"
string supported-structure
    uint32 supported-attribute-mask
    uint32 supported-attribute-bits
    uint32 supported-open-flags
    uint32 supported-access-mask
    uint32 max-read-size
    uint16 supported-open-block-vector
    uint16 supported-block-vector
    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 by the client to the 'File Attributes' valid-attribute-flags field (Section 7.1) in order to ensure that no unsupported attributes are present during a operation which writes attributes.

supported-attribute-bits

This mask MAY be applied by the client to the 'File Attributes' attrib-bits field (Section 7.9) in order to ensure that no unsupported attrib-bits are present during a operation which writes attribute bits.

supported-open-flags

The supported-open-flags mask MAY be applied by the client to the SSH_FXP_OPEN (Section 8.1.1) flags field.

supported-access-mask

This mask may be applied by the client to the ace-mask field of an ACL during a operation that writes the ACL.

This mask SHOULD NOT be applied to the desired-access field of the SSH_FXP_OPEN (Section 8.1.1) request. Doing so will simply result in not requesting the access required by the client. In the case of open operations, the server is responsible for translating the client's requested access to a mode it supports that is sufficient to grant all access requested by the client.

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 4.

supported-open-block-vector**supported-block-vector**

16-bit masks specifying which combinations of blocking flags are supported. Each bit corresponds to one combination of the SSH_FXF_BLOCK_READ, SSH_FXF_BLOCK_WRITE, SSH_FXF_BLOCK_DELETE, and SSH_FXF_BLOCK_ADVISORY bits from Section 8.1.1.3, with a set bit corresponding to a supported combination and a clear bit an unsupported combination. The index of a bit, bit zero being the least significant bit, viewed as a four-bit number, corresponds to a combination of flag bits, shifted right so that BLOCK_READ is the least significant bit. The combination 'no blocking flags' MUST be supported, so the low bit will always be set.

For example, a server supporting only the classic advisory read (shared) and write (exclusive) locks would set the bits corresponding to READ+WRITE+ADVISORY, 0b1011, and WRITE+ADVISORY, 0b1010, plus the always-set bit 0b0000, giving a mask value of 0b0000110000000001, or 0x0c01; a server supporting no locking at all would set only bit zero, giving 0x0001.

'supported-open-block-masks' applies to the SSH_FXP_OPEN (Section 8.1.1) flags field. 'supported-block-masks' applies to the SSH_FXF_BLOCK request.

attrib-extension-count

Count of extension names in the attrib-extension-names array.

attrib-extension-names

Names of extensions that can be used in an ATTRS (Section 7.14) structure.

extension-count

Count of extension names in the extension-names array.

extension-names

Names of extensions that can be used with the SSH_FXP_EXTEND (Section 10) packet.

Naturally, if a given attribute field, attribute mask bit, open flag, or extension is required for correct operation, the client **MUST** either not mask the bit 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.

The supported capabilities of the acl attribute are sent using the following extension.

```
string "acl-supported"
string supported-structure
uint32 capabilities
```

'capabilities' is a combination of the following bits:

SSH_ACL_CAP_ALLOW	0x00000001
SSH_ACL_CAP_DENY	0x00000002
SSH_ACL_CAP_AUDIT	0x00000004
SSH_ACL_CAP_ALARM	0x00000008
SSH_ACL_CAP_INHERIT_ACCESS	0x00000010
SSH_ACL_CAP_INHERIT_AUDIT_ALARM	0x00000020

```
SSH_ACL_CAP_ALLOW
SSH_ACL_CAP_DENY
SSH_ACL_CAP_AUDIT
SSH_ACL_CAP_ALARM
```

The server supports the associated ACL ACE type.

SSH_ACL_CAP_INHERIT_ACCESS

The server can control whether a ACL will inherit DENY and ALLOW ACEs that are marked inheritable from it's parent object.

SSH_ACL_CAP_INHERIT_AUDIT_ALARM

The server can control whether a ACL will inherit AUDIT or ALARM ACEs that are marked inheritable from it's parent object.

5.5. 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. Defined versions are: "2", "3", "4", "5", "6". Any other version advertised by the server must follow the DNS extensibility naming convention outlined in [RFC4251].

For example: "2,3,6,private@example.com".

If the client and server have negotiated a a version greater than or equal to 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"  
string 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.

Typically, the client SHOULD NOT down-grade the protocol version using this extension; however, it is not forbidden to do so. One reason a client might do so is to work around a buggy implementation.

6. 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 13) 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 preform a valid translation to UTF-8. When an invalid translation to UTF-8 is preformed, 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 it's 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 SSH_FX_OP_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.

7. 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
extension-pair	extensions	

7.1. 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 be added only 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_ACESSTIME	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.

7.2. 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.

7.3. 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.

If this field is present during file creation, it indicates the number of bytes the client intends to transfer, but SHOULD NOT affect the creation of the file. If the file is a text-file, then the bytes

are 'as encoded on the wire.' The server can use this information to determine if the client sent all the intended data and the file was transferred in its entirety.

If this field is present during a setstat operation, the file MUST be extended or truncated to the specified size. If the file is extended, the semantics are to write the byte value 0x00 from the previous end of file to the new offset.

Files opened with the SSH_FXF_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_TEXT flag.

7.4. 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 7.9). It will usually be larger than the 'size' field, however, as most file-systems allocate space to files in units that are larger than a single byte.

When present during file creation, the file SHOULD be created and the specified number of bytes preallocated. If the preallocation 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 allocation SHOULD be changed to the specified size. If the allocation size is increased, additional space is allocated to the file, but the 'size' is not changed (the end-of-file marker is not move.) If the new allocation size is smaller, than the operation is effectively a truncation.

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.

7.5. 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.

7.6. 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.

7.7. 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.

7.8. ACL

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

The structure of the ACL is:

```
uint32  acl-flags
uint32  ace-count
ACE     ace[ace-count]
```

The ACE data structure is composed as follows:

```
uint32  ace-type
uint32  ace-flag
uint32  ace-mask
string  who [UTF-8]
```

acl-flags

SFX_ACL_CONTROL_INCLUDED	0x00000001
SFX_ACL_CONTROL_PRESENT	0x00000002
SFX_ACL_CONTROL_INHERITED	0x00000004
SFX_ACL_AUDIT_ALARM_INCLUDED	0x00000010

SFX_ACL_AUDIT_ALARM_INHERITED 0x00000020

SFX_ACL_CONTROL_INCLUDED

SFX_ACL_CONTROL_PRESENT

SFX_ACL_CONTROL_INHERITED

If INCLUDED is set during a setstat operation, then the client intends to modify the ALLOWED/DENIED entries of the ACL. Otherwise, the client intends for these entries to be preserved.

If the PRESENT bit is not set, then the client wishes to remove control entries. If the server doesn't support separate control and audit information, the client MUST not clear this bit without also clearing the AUDIT_ALARM_PRESENT bit.

If the PRESENT bit is clear, then control of the file MAY be through the permissions mask. The server MAY also grant full access to the file.

If the both the INCLUDE and the PRESENT bit are set, but there are no ALLOW/DENY entries in the list, the client wishes to deny all access to the file or directory. The server may have to transform this into a ACL with a deny entry to implement it.

If INHERITED is set, then ALLOW/DENY ACEs MAY be inherited from the parent directory. If it is off, then they MUST not be INHERITED. If the server does not support controlling inheritance, then the client MUST clear this bit; in this case the inheritance properties of the server are undefined.

SFX_ACL_AUDIT_ALARM_INCLUDED

SFX_ACL_AUDIT_ALARM_INHERITED

If INCLUDE is set during a setstat operation, then the client intends to modify the AUDIT/ALARM entries of the ACL. Otherwise, the client intends for these entries to be preserved.

If INHERITED is set, then AUDIT/ALARM ACEs MAY be inherited from the parent directory. If it is off, then they MUST not be INHERITED. If the server does not support controlling inheritance, then the client MUST clear this bit; in this case the inheritance properties of the server are undefined.

Because some server require special permissions / privileges in order to modify the AUDIT/ALARM entries in the ACL, it is important to communicate to the server the clients intent to modify these entries. The client MUST both use the ACCESS_AUDIT_ALARM_ATTRIBUTES bit in the desired attribute of the

open request and must set the `SFX_ACL_AUDIT_ALARM_INCLUDED` during the `setstat` operation.

Clients that do not intend specifically to modify the `AUDIT` or `ALARM` entries SHOULD NOT set `SSH_FXF_ACCESS_AUDIT_ALARM_INFO` in the open-flags and SHOULD NOT set the `SFX_ACL_AUDIT_ALARM_INCLUDED` bit because these operations are often privileged and will fail.

If the `SFX_ACL_AUDIT_ALARM_INCLUDED` is set, and the requested change can not be made, the server MUST fail the request.

Servers that do not separate control and audit/alarm information may have to read the existing ACL and merge in entries not included by the client. The server must take this into account when opening files with the `ACE4_WRITE_ACL` permission requested.

ace-type

one of the following four values (taken from NFS Version 4 Protocol [RFC3010]):

<code>ACE4_ACCESS_ALLOWED_ACE_TYPE</code>	<code>0x00000000</code>
<code>ACE4_ACCESS_DENIED_ACE_TYPE</code>	<code>0x00000001</code>
<code>ACE4_SYSTEM_AUDIT_ACE_TYPE</code>	<code>0x00000002</code>
<code>ACE4_SYSTEM_ALARM_ACE_TYPE</code>	<code>0x00000003</code>

ace-flag

A combination of the following flag values. See NFS Version 4 Protocol [RFC3010] section 5.9.2:

<code>ACE4_FILE_INHERIT_ACE</code>	<code>0x00000001</code>
<code>ACE4_DIRECTORY_INHERIT_ACE</code>	<code>0x00000002</code>
<code>ACE4_NO_PROPAGATE_INHERIT_ACE</code>	<code>0x00000004</code>
<code>ACE4_INHERIT_ONLY_ACE</code>	<code>0x00000008</code>
<code>ACE4_SUCCESSFUL_ACCESS_ACE_FLAG</code>	<code>0x00000010</code>
<code>ACE4_FAILED_ACCESS_ACE_FLAG</code>	<code>0x00000020</code>
<code>ACE4_IDENTIFIER_GROUP</code>	<code>0x00000040</code>

ace-mask

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

UTF-8 string of the form described in 'Owner and Group' (Section 7.5)

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@.

7.9. 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 they know 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 5.4). 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_APPEND_DATA or the SSH_FXF_APPEND_DATA_ATOMIC flag (Section 8.1.1.3) 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.

7.10. 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_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_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_TEXT_MODE flag.

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

7.11. 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.

7.12. link-count

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

7.13. 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 7.9) bit in the `attrib-bits` field.

7.14. 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 5.4).

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 be defined only 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.

8. Requests From the Client to the Server

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

8.1. 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.

8.1.1. 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.)

8.1.1.1. 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.

8.1.1.2. desired-access

The 'desired-access' field is a bitmask containing a combination of values from the ace-mask flags (Section 7.8). 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.

8.1.1.3. 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_APPEND_DATA	= 0x00000008
SSH_FXF_APPEND_DATA_ATOMIC	= 0x00000010
SSH_FXF_TEXT_MODE	= 0x00000020
SSH_FXF_BLOCK_READ	= 0x00000040
SSH_FXF_BLOCK_WRITE	= 0x00000080
SSH_FXF_BLOCK_DELETE	= 0x00000100
SSH_FXF_BLOCK_ADVISORY	= 0x00000200
SSH_FXF_NOFOLLOW	= 0x00000400
SSH_FXF_DELETE_ON_CLOSE	= 0x00000800
SSH_FXF_ACCESS_AUDIT_ALARM_INFO	= 0x00001000
SSH_FXF_ACCESS_BACKUP	= 0x00002000
SSH_FXF_BACKUP_STREAM	= 0x00004000
SSH_FXF_OVERRIDE_OWNER	= 0x00008000

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_APPEND_DATA

Data is always written at the end of the file. The offset field of **SSH_FXP_WRITE** requests is 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_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_MODE

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 5.3))

When a file is opened with this 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_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.

`SSH_FXF_BLOCK_READ`

The server MUST guarantee that no other 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_BLOCK_WRITE`

The server MUST guarantee that no other handle 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_BLOCK_DELETE`

The server MUST guarantee that no other handle has been opened with `ACE4_DELETE` access, opened with the `SSH_FXF_DELETE_ON_CLOSE` flag set, and that no other handle will be opened with `ACE4_DELETE` access or with the `SSH_FXF_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_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_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_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.

It is implementation specific whether the directory entry is removed immediately or when the handle is closed.

`SSH_FXF_ACCESS_AUDIT_ALARM_INFO`

The client wishes the server to enable any privileges or extra capabilities that the user may have in to allow the reading and writing of AUDIT or ALARM access control entries.

SSH_FXF_ACCESS_BACKUP

The client wishes the server to enable any privileges or extra capabilities that the user may have in order to bypass normal access checks for the purpose of backing up or restoring files.

SSH_FXF_BACKUP_STREAM

This bit indicates that the client wishes to read or write a backup stream. A backup stream is a system dependent structured data stream that encodes all the information that must be preserved in order to restore the file from backup medium.

The only well defined use for backup stream data read in this fashion is to write it to the same server to a file also opened using the `BACKUP_STREAM` flag. However, if the server has a well defined backup stream format, there may be other uses for this data outside the scope of this protocol.

ACCESS_OVERRIDE_OWNER

This bit indicates that the client wishes the server to enable any privileges or extra capabilities that the user may have in order to gain access to the file with `WRITE_OWNER` permission.

This bit **MUST** always be specified in combination with `ACE4_WRITE_OWNER`.

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_APPEND_DATA and or SSH_FXF_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
```

8.1.2. 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).

8.1.3. 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.

8.2. Reading and Writing

8.2.1. 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

The offset (in bytes) relative to the beginning of the file that the read **MUST** start at. This field is ignored if `SSH_FXF_TEXT_MODE` 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 its 'supported2' (Section 5.4) extension and 'length' is \leq 'max-read-size', then failure to return 'length' bytes indicates that EOF or an error occurred.

8.2.2. 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.

8.2.3. 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

The offset (in bytes) relative to the beginning of the file that the write MUST start at. This field is ignored if SSH_FXF_TEXT_MODE 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.

8.3. 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.

If 'filename' is a symbolic link, the link is removed, not the file it points to.

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_FXF_RENAME_OVERWRITE 0x00000001
SSH_FXF_RENAME_ATOMIC    0x00000002
SSH_FXF_RENAME_NATIVE     0x00000004
```

If the server cannot support the requested mode of operation, it must return `SSH_FX_OP_UNSUPPORTED`.

If flags does not include `SSH_FXF_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_FXF_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_FXF_RENAME_ATOMIC` implies `SSH_FXF_RENAME_OVERWRITE`.

If '`SSH_FXF_RENAME_OVERWRITE`' is not present, and the server does not support this mode natively, it MAY emulate it by checking for the existence of a file before executing the rename operation.

If the '`SSH_FXF_RENAME_ATOMIC`' is specified without the the '`SSH_FXF_RENAME_OVERWRITE`', then the server MUST be able to perform the check and rename operation atomically.

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

If flags includes `SSH_FXF_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.

8.4. 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.

8.5. 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.

8.6. 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.

8.7. 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.

target-path

Specifies the path of a target object to which the newly created link will refer. In the case of a symbolic link, this path may not exist.

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.

8.8. Byte-range locks

8.8.1. Obtaining a byte range lock

`SSH_FXP_BLOCK` 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.

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

handle

'handle' is a handle returned by SSH_FXP_OPEN or SSH_FXP_OPENDIR.

Note that some servers MAY return SSH_FX_OP_UNSUPPORTED if the handle is a directory 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 current end-of-file. (This operation is equivalent to doing a SSH_FXP_STAT to retrieve the 'size' and then using that to calculate the length.)

uLockMask

A bit mask of SSH_FXF_BLOCK_* values; the meanings are described in Section 8.1.1.3.

The result is a SSH_FXP_STATUS packet. If the requested range overlaps in any fashion with an already granted SSH_FXP_BLOCK request (even if the request is from the same client in the same connection), the result is SSH_FX_BYTE_RANGE_LOCK_REFUSED.

It is permissible for 'offset' to be \geq 'size', or for 'length' to extend past 'size'. However, in no case does this operation effect 'size'. Naturally, all of the same restrictions regarding conflicting locks / access apply to such a lock.

Take together the 'offset' and the 'length' can be considered to form a unique key, which is then used to release the lock using the SSH_FXP_UNBLOCK (below) request.

8.8.2. Releasing a byte range lock

SSH_FXP_UNBLOCK removes a previously acquired byte-range lock on the specified handle. The 'offset' and 'length' MUST match exactly the offset and length specified to the SSH_FXP_LOCK request.

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

handle

'handle' on which a SSH_FXP_BLOCK request has previously been issued.

offset

Beginning of the byte-range to unlock-- must match exactly the 'offset' given to the SSH_FXP_BLOCK request.

length

Number of bytes in the range to unlock. The special value 0 means unlock from 'offset' to the end of the file. This must match exactly the 'length' given to the SSH_FXP_BLOCK request.

The result is a SSH_FXP_STATUS packet. If the 'offset' and 'length' do not exactly match a previously granted range, the server MUST return SSH_FX_NO_MATCHING_BYTE_RANGE_LOCK.

8.9. 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]
byte    control-byte [optional]
string  compose-path[0..n] [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.

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. In addition, the path MUST NOT resolve symbolic links. This allows paths to be composed for the `SSH_FXP_REMOVE` command to remove symbolic links.

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.

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.

The client may specify multiple 'compose-path' elements, in which case the server should build the resultant path up by applying each compose path to the accumulated result until all 'compose-path' elements have been applied.

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.

8.9.1. 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.

9. 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.

9.1. 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.

Implementations MUST be prepared to receive unexpected error codes and handle them sensibly (such as by treating them as equivalent to SSH_FX_FAILURE). Future protocol revisions may add additional error codes without changing the version number.

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_OWNER_INVALID	29	
SSH_FX_GROUP_INVALID	30	
SSH_FX_NO_MATCHING_BYTE_RANGE_LOCK	31	

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 insufficient 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 principal 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 or, an SSH_FXF_NOFOLLOW open encountered a symbolic link as the final component

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

An read or write operation failed because another process's mandatory 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.

SSH_FX_OWNER_INVALID

The principal specified can not be assigned as an owner of a file.

SSH_FX_GROUP_INVALID

The principal specified can not be assigned as the primary group of a file.

SSH_FX_NO_MATCHING_BYTE_RANGE_LOCK

The requested operation could not be completed because the specied byte range lock has not been granted.

9.2. 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.

9.3. 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 8.2.1.)

end-of-file

This field is optional, and if present and true 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.)

9.4. 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

If this field is present and true, there are no more entries to be read. This field should either be omitted or be true unless the request is SSH_FXP_READDIR.

9.5. 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".

10. 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, following the the DNS extensibility naming convention outlined in [RFC4251], or defined by IETF consensus.

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.

11. 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 4.1).)

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

12. IANA Considerations

IANA registries need to be created for each of the following:

- o The packet types defined in Section 4.3
- o The extension specified in this draft, which are: 'text-seek', 'supported2', 'acl-supported', 'newline', 'versions', 'version-select', 'filename-charset', 'filename-translation-control'

13. 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 may be able to manipulate files on the server 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 [RFC4252]).

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 7.6) 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.

14. Changes from Previous Protocol Versions

RFC EDITOR: PLEASE REMOVE ENTIRE SECTION BEFORE PUBLISHING

Please refer to the following web page for pervious versions of the protocol:

<http://tools.ietf.org/wg/secsh/draft-ietf-secsh-filexfer/>

RFC EDITOR: END PLEASE REMOVE ENTIRE SECTION BEFORE PUBLISHING

15. References

15.1. Normative References

- [RFC3010] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "NFS version 4 Protocol", RFC 3010, December 2000.
- [RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.
- [RFC4253] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.
- [RFC4254] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Connection Protocol", RFC 4254, January 2006.
- [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.

15.2. 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.
- [RFC4252] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

Trademark notice

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

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

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.

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 (2006). 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.

