Secure Shell Working Group
Internet-Draft
Expires: April 26, 2005

J. Galbraith
VanDyke Software
October 26, 2004

SSH File Transfer Protocol
draft-ietf-secsh-filexfer-06.txt

Status of this Memo

Copyright Notice

Abstract

The SSH File Transfer Protocol provides secure file transfer functionality over any reliable data stream. It is the standard file transfer protocol for use with the SSH2 protocol. This document describes the file transfer protocol and its interface to the SSH2 protocol suite.

Table of Contents

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 the SSH2 protocol [1]. and that the server has already authenticated the client, and that the identity of the client user is available to the protocol.

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

The packet format descriptions in this specification follow the notation presented in the secsh architecture draft. [1]

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 RFC 2246 [7] and transfer of management information in VPN applications.

2. Use with the SSH Connection Protocol

When used with the SSH2 Protocol suite, this protocol is intended to be used from the SSH Connection Protocol [3] as a subsystem, as described in section ''Starting a Shell or a Command''. The subsystem name used with this protocol is "sftp".

2.1 The Use of 'stderr' in the server

This protocol uses stdout and stdin to transmit binary protocol data. The "session" channel SSH Connection Protocol [3], which is used by the subsystem, also supports the use of stderr.

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

For example, during initialization, there is no client request active, so errors or warning information cannot be sent to the client as part of the SFTP protocol at this early stage. However, the errors or warnings MAY be sent as stderr text.

3. General Packet Format

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

```
uint32           length
byte             type
uint32           request-id
    ... type specific fields...
byte[length] data payload
```

'length' is 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 the secsh transport. [2]

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

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 us a monotonically increasing request sequence number (modulo 2^32). There is, however, no particular requirement the 'request-id' fields be unique.

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

3.1 Request Synchronization and Reordering

The protocol and implementations MUST process requests relating to

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

file transfer connections. The server may interleave and parallelize them at will.

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

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

This document defines one data type in addition to those defined in secsh architecture draft. [1]

int64
: Represents a 64-bit signed integer. Stored as eight bytes in the order of decreasing significance (network byte order).

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.

3.2 Packet Types

The following values are defined for packet types.

```
      #define SSH_FXP_INIT               1
      #define SSH_FXP_VERSION            2
      #define SSH_FXP_OPEN               3
      #define SSH_FXP_CLOSE              4
      #define SSH_FXP_READ               5
      #define SSH_FXP_WRITE              6
      #define SSH_FXP_LSTAT              7
      #define SSH_FXP_FSTAT              8
      #define SSH_FXP_SETSTAT            9
      #define SSH_FXP_FSETSTAT          10
      #define SSH_FXP_OPENDIR           11
      #define SSH_FXP_READDIR           12
      #define SSH_FXP_REMOVE            13
      #define SSH_FXP_MKDIR             14
      #define SSH_FXP_RMDIR             15
      #define SSH_FXP_REALPATH          16
      #define SSH_FXP_STAT              17
      #define SSH_FXP_RENAME            18
      #define SSH_FXP_READLINK          19
      #define SSH_FXP_SYMLINK           20


      #define SSH_FXP_STATUS           101
      #define SSH_FXP_HANDLE           102
      #define SSH_FXP_DATA             103
      #define SSH_FXP_NAME             104
      #define SSH_FXP_ATTRS            105


      #define SSH_FXP_EXTENDED         200
      #define SSH_FXP_EXTENDED_REPLY   201


      RESERVED_FOR_EXTENSIONS          210-255
```

Additional packet types should only be defined if the protocol version number (see Section ''Protocol Initialization'') is incremented, and their use MUST be negotiated using the version number. However, the 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.

4. Protocol Initialization

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

```
**********************  DO NOT IMPLEMENT ***********************
**********************  DO NOT IMPLEMENT ***********************
*****                                                      *****
*****      There will be more edits after IETF 61.  *****
*****                                                      *****
**********************  DO NOT IMPLEMENT ***********************
**********************  DO NOT IMPLEMENT ***********************
```

4.1 Client Initialization

The SSH_FXP_INIT packet (from client to server) has the following data:

```
    uint32 version
```

Version 3 of this protocol allowed clients to include extensions in the SSH_FXP_INIT packet; however, this can cause interoperability problems with version 1 and version 2 servers because the client must send this packet before knowing the servers version.

In this version of the protocol, clients MUST use the SSH_FXP_EXTENDED packet to send extensions to the server after version exchange has completed. Clients MUST NOT include extensions in the version packet. This will prevent interoperability problems with older servers

4.2 Server Initialization

The SSH_FXP_VERSION packet (from server to client) has the following data:

```
uint32 version
<extension data>
```

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

The extension data may be empty, or may be a sequence of

```
string extension_name
string extension_data
```

pairs (both strings MUST always be present if one is, but the 'extension_data' string may be of zero length). If present, these strings indicate extensions to the baseline protocol. The 'extension_name' field(s) identify the name of the extension. The name should be of the form "name@domain", where the domain is the DNS domain name of the organization defining the extension. Additional names that are not of this format may be defined later by the IETF. Implementations MUST silently ignore any extensions whose name they do not recognize.

4.3 Determining Server Newline Convention

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

The SSH_FXF_TEXT file open flag (Section 7.1.1) makes it possible to request that the server translate a file to a 'canonical' wire format. This format uses \r\n as the line separator.

Servers for systems using multiple newline characters (for example, Mac OS X or VMS) or systems using counted records, MUST translate to 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 "\r" or "\n" or "\r\n")
```

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 "\r\n" sequence, a lone "\r" or a lone "\n" SHOULD be written through without change.

4.4 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. In order to facilitate 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, the following extension is introduced.

```
    string "supported"
    string supported-structure
        uint32 supported-attribute-mask
        uint32 supported-attribute-bits
        uint32 supported-open-flags
        uint32 supported-access-mask
        uint32 max-read-size
        string extension-names[0..n]
```

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

supported-attribute-bits
: This mask MAY by applied to the 'File Attributes' attrib-bits field (Section 6.9) to ensure that no unsupported attrib-bits are present during a operation which writes attributes.

supported-open-flags
: The supported-open-flags mask MAY be applied to the SSH_FXP_OPEN (Section 7.1.1) flags field.

supported-access-mask
: The supported-access-mask mask MAY be applied to the SSH_FXP_OPEN (Section 7.1.1) desired-access field or the ace-mask field of an ACL.

max-read-size
   This is the maximum read size that the server gaurantees to complete. For example, certain embedded server implementations only complete 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, it MUST return at least the max-read-size number of bytes for any read requesting max-read-size bytes. Failure to return max-read-size bytes in such a case indicates either EOF or another error condition occurred.

extension names
The extension names may be empty (contains zero strings), or it may contain any named extensions that the server wishes to advertise.

The client must be able to differentiate between attribute extensions (Section 6.13) and extended requests (Section 9) by the extension name.

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

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

4.5 Version re-negotiation

The version exchange during protocol startup forces an implementation to support all versions up to it's highest supported version; however, there have been a number of SFTP protocol versions deployed, and it is supposed that more implementations will support the final version of this protocol if they don't have to support all versions between their currently deployed version and the final version. Furthermore, only the current version of this protocol is documented, so supporting earlier versions becomes problematic.

Therefore, the server SHOULD send the following extension as part of it's INIT packet to inform the client of the versions it supports.

```
    string "versions"
    string comma-seperated-versions
```

'comma-seperated-versions' is a string of comma seperated version numbers, for example, "3,6,7"

A client wishing to support two non-continigous version of the protocol must negotiate the lowest version for which it supports all previous versions. When the client recieves the servers INIT packet, if it includes the "versions" extension, it MAY send the following extended request:

```
byte   SSH_FXP_EXTENDED
uint32 request-id
string "version"
uint32 version-from-list
```

If the 'version-from-list' is one of the versions on the servers list, the server MUST respond with SSH_FX_OK. If the server did not send the "versions" extension, or the version-from-list was not included, the server MAY send a status response describing the failure, but MUST then close the channel.

Although this request does take a full round trip, no client need wait for the response before continuing, because any valid request MUST succeed.

5. File Names

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

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

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

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

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

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

The prefered encoding for filenames is UTF-8. This is consistant with IETF Policy on Character Sets and Languages [8] 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.

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.

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 reversable. 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 it's 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 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 or disable filename translation according to the value of 'do-translate'

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

6. File Attributes

A new compound data type 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                 present only if flag SIZE
uint64   allocation-size      present only if flag ALLOCATION_SIZE
string   owner                present only if flag OWNERGROUP
string   group                present only if flag OWNERGROUP
uint32   permissions          present only if flag PERMISSIONS
int64    atime                present only if flag ACCESSTIME
uint32   atime_nseconds       present only if flag SUBSECOND_TIMES
int64    createtime           present only if flag CREATETIME
uint32   createtime_nseconds  present only if flag SUBSECOND_TIMES
int64    mtime                present only if flag MODIFYTIME
uint32   mtime_nseconds       present only if flag SUBSECOND_TIMES
string   acl                  present only if flag ACL
uint32   attrib-bits          present only if flag BITS
byte     text-hint            present only if flag TEXT_HINT
string   mime-type            present only if flag MIME_TYPE
uint32   link-count           present only if flag LINK_COUNT
string   untranslated-name    present only if flag UNTRANSLATED_NAME
uint32   extended_count       present only if flag EXTENDED
string   extended_type
string   extended_data
...      more extended data (extended_type - extended_data pairs),
         so that number of pairs equals extended_count
```

6.1 valid-attribute-flags

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

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

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

In general, unless otherwise specified, if a server cannot support writing an attribute requested, it must fail the setstat operation.

In this case, none of the attributes SHOULD be changed.

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

The following values are defined:

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

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

6.2 Type

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

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

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

6.3 Size

The 'size' field specifies the number of bytes that can be read from the file, or in other words, the location of the end-of-file. If it is present during file creation, the file MUST be created and then the EOF set to 'size'. A read from such a file SHOULD return nul bytes, but this is not required if the underlying filesystem has different characteristics.

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

Files opened with the SSH_FXF_TEXT flag may have a size that is greater or less than the value of the size field.

6.4 AllocationSize

The 'allocation-size' field specifies the number of bytes that the file consumes on disk. This is normally greater than or equal to the 'size' field. If it is present during file creation, it should be treated as a hint as to the eventual file size. The server MAY choose to preallocate the disk space to save the overhead of repeated extends. However, the file size MUST NOT be set to this value. In other words, a read from such a file MUST fail with an EOF error. (Unless 'size' was also set.)

If the server is unable to honor this hint during create, the create should succeed regardless. Because this field is a hint, the field may be specified even if the server doesn't set the bit in it's supported-attribute-mask.

If this field is present during a setstat operation, the file SHOULD be extended or truncated to the specified size. Clients SHOULD therefore be careful specifying size during a setstat operation.

If the file is extended by this operation, 'size' MUST not be affected. If the file is truncated by this operation, 'size' will be changed ot match the new file allocation.

If a server can not honor the setstat operation, it MUST NOT set allocation-size in it's supported-attribute-mask, though it MAY still send the allocation-size data if it can retrieve it. In addition, such a server MUST fail a setstat operaiton that has the allocation-size field present.

6.5 Owner and Group

The 'owner' and 'group' fields are represented as UTF-8 strings; this is the form used by NFS v4. See NFS version 4 Protocol [4]. 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 with 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.

6.6 Permissions

The 'permissions' field contains a bit mask specifying file permissions. These permissions correspond to the st_mode field of the stat structure defined by POSIX [5].

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

```
#define S_IRUSR  0000400 (octal)
#define S_IWUSR  0000200
#define S_IXUSR  0000100
#define S_IRGRP  0000040
```

```
#define S_IWGRP  0000020
#define S_IXGRP  0000010
#define S_IROTH  0000004
#define S_IWOTH  0000002
#define S_IXOTH  0000001
#define S_ISUID  0004000
#define S_ISGID  0002000
#define S_ISVTX  0001000
```

Implementations MUST NOT send bits that are not defined.

6.7 Times

The 'atime', 'createtime', and 'mtime' contain the accesses, creation, and modification times of the files, respectively. They are represented as seconds from Jan 1, 1970 in UTC.

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

6.8 ACL

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

```
uint32   ace-count

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

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

```
#define ACE4_ACCESS_ALLOWED_ACE_TYPE 0x00000000;
#define ACE4_ACCESS_DENIED_ACE_TYPE  0x00000001;
#define ACE4_SYSTEM_AUDIT_ACE_TYPE   0x00000002;
#define ACE4_SYSTEM_ALARM_ACE_TYPE   0x00000003;
```

ace-flag is a combination of the following flag values. See NFS

Version 4 Protocol [4] section 5.9.2:

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

ace-mask is any combination of the following flags (taken from NFS Version 4 Protocol [4] section 5.9.3:

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

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

Also, as per '5.9.4 ACE who' [4] there are several identifiers that need to be understood universally. Some of these identifiers cannot be understood when an client access the server, but have meaning when a local process accesses the file. The ability to display and modify these permissions is permitted over SFTP.

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

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

6.9 attrib-bits

These bits reflect various attributes of the file or directory on the server.

The following attrib-bits are defined:

```
#define SSH_FILEXFER_ATTR_FLAGS_READONLY         0x00000001
#define SSH_FILEXFER_ATTR_FLAGS_SYSTEM           0x00000002
#define SSH_FILEXFER_ATTR_FLAGS_HIDDEN           0x00000004
#define SSH_FILEXFER_ATTR_FLAGS_CASE_INSENSITIVE 0x00000008
#define SSH_FILEXFER_ATTR_FLAGS_ARCHIVE          0x00000010
#define SSH_FILEXFER_ATTR_FLAGS_ENCRYPTED        0x00000020
#define SSH_FILEXFER_ATTR_FLAGS_COMPRESSED       0x00000040
#define SSH_FILEXFER_ATTR_FLAGS_SPARSE           0x00000080
#define SSH_FILEXFER_ATTR_FLAGS_APPEND_ONLY      0x00000100
#define SSH_FILEXFER_ATTR_FLAGS_IMMUTABLE        0x00000200
#define SSH_FILEXFER_ATTR_FLAGS_SYNC             0x00000400
#define 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 operating system.

SSH_FILEXFER_ATTR_FLAGS_HIDDEN
File SHOULD NOT be shown to user unless specifically requested. For example, most UNIX systems SHOULD set this bit if the filename begins with a 'period'. This bit may be read-only (Section 4.4). Most UNIX systems will not allow this to be changed.

SSH_FILEXFER_ATTR_FLAGS_CASE_INSENSITIVE
This attribute can only apply 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 server 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
The file can only be opened for writing in append mode.

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

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

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

SSH_FILEXFER_ATTR_FLAGS_TRANSLATION_ERR

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

6.10 Text Hint

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

```
#define SSH_FILEXFER_ATTR_KNOWN_TEXT        0x01
#define SSH_FILEXFER_ATTR_GUESSED_TEXT      0x01
#define SSH_FILEXFER_ATTR_KNOWN_BINARY      0x01
#define SSH_FILEXFER_ATTR_GUESSED_BINARY    0x01
```

SSH_FILEXFER_ATTR_KNOWN_TEXT
: The server knows the file is a text file, and should be opened using the SSH_FXF_ACCESS_TEXT_MODE flag.

SSH_FILEXFER_ATTR_GUESSED_TEXT
: The server has applied a hueristic or other mechanism and believes that the file should be opened with the SSH_FXF_ACCESS_TEXT_MODE flag.

SSH_FILEXFER_ATTR_KNOWN_BINARY
: The server knows the file has binary content.

SSH_FILEXFER_ATTR_GUESSED_BINARY
: The server has applied a hueristic or other mechanism and believes has binary content, and should not be opened with the SSH_FXF_ACCESS_TEXT_MODE flag.

This flag MUST NOT be present during a setstat operation. If this flag is present during an fsetstat operation, the file handle is converted to a text-mode handle, as if it had been opened with SSH_FXF_ACCESS_TEXT_MODE.

6.11 Mime type

The 'mime-type' field contains the mime-type [9] string. Most servers will not know this information and should not set the bit in their supported-attribute-mask.

6.12 Link Count

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

6.13 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 extended_type-extended_data pairs that follow. Each of these pairs specifies an extended attribute. For each of the attributes, the extended_type field should be a string of the format "name@domain", where "domain" is a valid, registered domain name and "name" identifies the method. The IETF may later standardize certain names that deviate from this format (e.g., that do not contain the "@" sign). The interpretation of 'extended_data' depends on the type. Implementations SHOULD ignore extended data fields that they do not understand.

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

7. Requests From the Client to the Server

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

7.1 Opening and Closing Files and Directories

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

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

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

7.1.1 Opening a File

Files are opened and created using the SSH_FXP_OPEN message:

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

The response to this message will be either SSH_FXP_HANDLE (if the operation is successful) or SSH_FXP_STATUS (if the operation fails).

The 'request-id' field is the request identifier as for all requests.

The 'filename' field specifies the file name. See Section ''File Names'' for more information.

The 'desired-access' field is a bitmask containing a combination of values from the ace-mask flags from section 5.7.

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

desired.

The following 'flags' are defined:

```
    SSH_FXF_ACCESS_DISPOSITION        = 0x00000007
        SSH_FXF_CREATE_NEW            = 0x00000000
        SSH_FXF_CREATE_TRUNCATE       = 0x00000001
        SSH_FXF_OPEN_EXISTING         = 0x00000002
        SSH_FXF_OPEN_OR_CREATE        = 0x00000003
        SSH_FXF_TRUNCATE_EXISTING     = 0x00000004
    SSH_FXF_ACCESS_APPEND_DATA        = 0x00000008
    SSH_FXF_ACCESS_APPEND_DATA_ATOMIC = 0x00000010
    SSH_FXF_ACCESS_TEXT_MODE          = 0x00000020
    SSH_FXF_ACCESS_READ_LOCK          = 0x00000040
    SSH_FXF_ACCESS_WRITE_LOCK         = 0x00000080
    SSH_FXF_ACCESS_DELETE_LOCK        = 0x00000100
    SSH_FXF_NOFOLLOW                  = 0x00000200
```

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 truncated.

SSH_FXF_OPEN_EXISTING

An existing file is opened. If the file does not exist, the server MUST return SSH_FX_NO_SUCH_FILE. If a directory in the path does not exist, the server SHOULD return SSH_FX_NO_SUCH_PATH. It is also acceptable if the server returns SSH_FX_NO_SUCH_FILE in this case.

SSH_FXF_OPEN_OR_CREATE

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

SSH_FXF_TRUNCATE_EXISTING

An existing file is opened and truncated. If the file does not exist, the server MUST return the same error codes as defined for SSH_FXF_OPEN_EXISTING.

SSH_FXF_ACCESS_APPEND_DATA
Data is always written at the end of the file. The offset field of the SSH_FXP_WRITE requests are ignored.

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

SSH_FXF_ACCESS_APPEND_DATA_ATOMIC
Data is always written at the end of the file. The offset field of the SSH_FXP_WRITE requests are ignored.

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

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

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

When a file is opened with the FXF_TEXT flag, the offset field in both the read and write function are ignored.

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

Clients SHOULD use the SSH_FXF_ACCESS_APPEND_DATA flag to append data to a text file rather then using write with a calculated offset.

To support seeks on text files the following SSH_FXP_EXTENDED packet is defined.

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

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

The response to a "text-seek" request is an SSH_FXP_STATUS message.

An attempt to seek past the end-of-file should result in a SSH_FX_EOF status.

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

Clients MUST be prepared to handle SSH_FX_OP_UNSUPPORTED returned for read or write operations that are not sequential.

SSH_FXF_ACCESS_READ_LOCK
The file should be opened with a read lock. The server MUST gaurantee that the client will be the exclusive reader of the file 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 gaurantee, it MUST return SSH_FX_OP_UNSUPPORTED.

SSH_FXF_ACCESS_WRITE_LOCK
The file should be opened with a write lock. The server MUST gaurantee that the client will be the exclusive writer of the file until the client closes the handle.

SSH_FXF_ACCESS_DELETE_LOCK
The file should be opened with a delete lock. The server MUST gaurantee that the file will not be deleted until the client closes the handle.

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.

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 exiting file is opened.

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

```
O_RDONLY
   desired-access = READ_DATA|READ_ATTRIBUTES

O_WRONLY
   desired-access = WRITE_DATA|WRITE_ATTRIBUTES

O_RDWR
   desired-access =
   READ_DATA|READ_ATTRIBUTES|WRITE_DATA|WRITE_ATTRIBUTES

O_APPEND
   desired-access = WRITE_DATA|WRITE_ATTRIBUTES|APPEND_DATA
   flags          = SSH_FXF_ACCESS_APPEND_DATA and or
   SSH_FXF_ACCESS_APPEND_DATA_ATOMIC

O_CREAT
   flags = SSH_FXF_OPEN_OR_CREATE

O_TRUNC
   flags = SSH_FXF_TRUNCATE_EXISTING

O_TRUNC|O_CREATE
   flags = SSH_FXF_CREATE_TRUNCATE
```

7.1.2 Opening a Directory

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

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

'request-id' is the request identifier.

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

file names.

The response to this message will be either SSH_FXP_HANDLE (if the operation is successful) or SSH_FXP_STATUS (if the operation fails).

7.1.3 Closing Handles

A handle is closed using the following request.

```
    byte   SSH_FXP_CLOSE
```

```
    uint32 request-id
    string handle
```

'request-id' is the request identifier, and '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.

7.2 Reading and Writing

7.2.1 Reading Files

The following request can be used to read file data:

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

where 'request-id' is the request identifier, 'handle' is an open file handle returned by SSH_FXP_OPEN, 'offset' is the offset (in bytes) relative to the beginning of the file from where to start reading, and 'length' is the maximum number of bytes to read.

In response to this request, the server will read as many bytes as it can from the file (up to 'length'), and return them in a SSH_FXP_DATA message. If an error occurs or EOF is encountered before reading any data, the server will respond with SSH_FXP_STATUS.

For normal disk files, it is normally guaranteed that this will read the specified number of bytes, or up to end of file. However, if the read length is very long, the server may truncate it if it doesn't support packets of that length. See General Packet Format (Section 3).

7.2.2 Reading Directories

In order to retrieve a directory listing, the client issues one or more SSH_FXP_READDIR requests. In order to obtain a complete directory listing, the client MUST issue repeated SSH_FXP_READDIR requests until the server responds with an SSH_FXP_STATUS message.

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

where 'request-id' is the request identifier, and 'handle' is a handle returned by SSH_FXP_OPENDIR. (It is a protocol error to attempt to use an ordinary file handle returned by SSH_FXP_OPEN.)

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

If there are no more names available to be read, the server MUST respond with a SSH_FXP_STATUS message with error code of SSH_FX_EOF.

7.2.3 Writing Files

Writing to a file is achieved using the following message:

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

where 'request-id' is a request identifier, 'handle' is a file handle returned by SSH_FXP_OPEN, 'offset' is the offset (in bytes) from the beginning of the file where to start writing, and 'data' is the data to be written.

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 zeroes 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.

The server responds to a write request with a SSH_FXP_STATUS message.

7.3 Removing and Renaming Files

The following request can be used to remove a file:

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

where 'request-id' is the request identifier and 'filename' is the name of the file to be removed. See Section ''File Names'' for more information. This request cannot be used to remove directories.

The server will respond to this request with a SSH_FXP_STATUS message.

Files (and directories) can be renamed using the SSH_FXP_RENAME message.

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

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

'flags' is 0 or a combination of:

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

If flags does not include SSH_FXP_RENAME_OVERWRITE, and there already exists a file with the name specified by newpath, the server MUST respond with SSH_FX_FILE_ALREADY_EXISTS.

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

If flags includes SSH_FXP_RENAME_ATOMIC and the server cannot replace the destination in an atomic fashion, then the server MUST respond with SSH_FX_OP_UNSUPPORTED.

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

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

requirements.

The server will respond to this request with a SSH_FXP_STATUS message.

7.4 Creating and Deleting Directories

New directories can be created using the SSH_FXP_MKDIR request. It has the following format:

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

where 'request-id' is the request identifier.

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

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

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

Directories can be removed using the SSH_FXP_RMDIR request, which has the following format:

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

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

The server responds to this request with a SSH_FXP_STATUS message.

7.5 Retrieving File Attributes

Very often, file attributes are automatically returned by SSH_FXP_READDIR. However, sometimes there is need to specifically retrieve the attributes for a named file. This can be done using the SSH_FXP_STAT, SSH_FXP_LSTAT and SSH_FXP_FSTAT requests.

SSH_FXP_STAT and SSH_FXP_LSTAT only differ in that SSH_FXP_STAT follows symbolic links on the server, whereas SSH_FXP_LSTAT does not follow symbolic links. Both have the same format:

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

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

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

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

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

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

where 'request-id' is the request identifier and 'handle' is a file handle returned by SSH_FXP_OPEN. The server responds to this request with SSH_FXP_ATTRS or SSH_FXP_STATUS.

7.6 Setting File Attributes

File attributes may be modified using the SSH_FXP_SETSTAT and SSH_FXP_FSETSTAT requests.

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

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

request-id
The request identifier to be returned as part of the response.

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
The handle is a handle previously returned from a SSH_FXP_OPEN request which identifies the file whose attributes are to be modified. 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 be have been successfully modified. If possible, servers SHOULD avoid this situation; however, client MUST be aware that this is possible.

7.7 Dealing with Symbolic 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_SYMLINK request creates a symbolic link on the server.

```
byte   SSH_FXP_SYMLINK
uint32 request-id
string linkpath   [UTF-8]
string targetpath [UTF-8]
```

where 'request-id' is the request identifier, 'linkpath' specifies the path name of the symlink to be created and 'targetpath' specifies the target of the symlink. The server shall respond with a SSH_FXP_STATUS.

7.8 Canonicalizing the Server-Side Path Name

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

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

where 'request-id' is the request identifier and 'path' specifies the path name to be canonicalized. The server will respond with a SSH_FXP_NAME packet containing the name in canonical form and a dummy attributes value. If an error occurs, the server may also respond with SSH_FXP_STATUS.

The server SHOULD fail the request if the path is not present on the server.

7.8.1 Best Practice for Dealing with Paths

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 use 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.

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 "c:/x/y/z/..".

As a second example, if the client wishes to open the file "x.txt" in the current directory, and server has returned "dka100:/x/y/z" as the

canonical path of the directory, the client SHOULD open
"dka100:/x/y/z/x.txt"

8. Responses from the Server to the Client

The server responds to the client using one of a few response packets. All requests can return a SSH_FXP_STATUS response upon failure. When the operation is successful, and no data needs to be returned, the SSH_FXP_STATUS response with SSH_FX_OK status is appropriate.

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

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

The format of the data portion of the SSH_FXP_STATUS response is as follows:

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

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

error/status code
Machine readable status code indicating the result of the request. Error code values are defined below. The value SSH_FX_OK indicates success, and all other values indicate failure.

error message

Human readable description of the error. 'language tag' 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:

```
    #define SSH_FX_OK                            0
    #define SSH_FX_EOF                           1
    #define SSH_FX_NO_SUCH_FILE                  2
    #define SSH_FX_PERMISSION_DENIED             3
    #define SSH_FX_FAILURE                       4
    #define SSH_FX_BAD_MESSAGE                   5
    #define SSH_FX_NO_CONNECTION                 6
    #define SSH_FX_CONNECTION_LOST               7
    #define SSH_FX_OP_UNSUPPORTED                8
    #define SSH_FX_INVALID_HANDLE                9
    #define SSH_FX_NO_SUCH_PATH                  10
    #define SSH_FX_FILE_ALREADY_EXISTS           11
    #define SSH_FX_WRITE_PROTECT                 12
    #define SSH_FX_NO_MEDIA                      13
    #define SSH_FX_NO_SPACE_ON_FILESYSTEM        14
    #define SSH_FX_QUOTA_EXCEEDED                15
    #define SSH_FX_UNKNOWN_PRINCIPLE             16
    #define SSH_FX_LOCK_CONFlICT                 17
    #define SSH_FX_DIR_NOT_EMPTY                 18
    #define SSH_FX_NOT_A_DIRECTORY               19
    #define SSH_FX_INVALID_FILENAME              20
    #define SSH_FX_LINK_LOOP                     21
```

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 occured, 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 can only be generated locally, and MUST NOT be return by a server.

SSH_FX_CONNECTION_LOST
The connection to the server was lost. This error can only be generated locally, and MUST NOT be return by a server.

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

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

SSH_FX_INVALID_HANDLE
The handle value was invalid.

SSH_FX_NO_SUCH_PATH
The file path does not exist or is invalid.

SSH_FX_FILE_ALREADY_EXISTS
The file already exists.

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

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

SSH_FX_NO_SPACE_ON_FILESYSTEM
The requested operation cannot be completed because there is no

free space on the filesystem.

SSH_FX_QUOTA_EXCEEDED
The operation cannot be completed because the it would exceed the users storage quota.

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

```
string unknown-name
```

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

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

SSH_FX_DIR_NOT_EMPTY
The directory is not empty.

SSH_FX_NOT_A_DIRECTORY
The specified file is not a directory.

SSH_FX_INVALID_FILENAME
The filename is not valid.

SSH_FX_LINK_LOOP
Too many symbolic links encountered.

The SSH_FXP_HANDLE response has the following format:

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

where 'request-id' is the request identifier, and 'handle' is 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.

The SSH_FXP_DATA response has the following format:

```
byte   SSH_FXP_DATA
uint32 request-id
string data
```

where 'request-id' is the request identifier, and '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 if end of file is reached or if the

read is from something other than a regular file.

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
```

where 'request-id' is the request identifier, 'count' is the number of names returned in this response, and the remaining fields repeat 'count' times. In the repeated part, 'filename' is 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), and 'attrs' is the attributes of the file as described in Section ''File Attributes''.

The SSH_FXP_ATTRS response has the following format:

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

where 'request-id' is the request identifier, and 'attrs' is the returned file attributes as described in Section ''File Attributes''.

9. Extensions

The SSH_FXP_EXTENDED request provides a generic extension mechanism for adding additional commands.

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

request-id
: Identifier to be returned from the server with the response.

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

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

request-specific data
: The rest of the request is defined by the extension, and servers should only attempt to interpret it if they 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 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 additional to any other data.

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

9.1 Checking File Contents

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

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

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

file-handle
Used if "md5-hash-handle" is specified; specifies a file handle to read the data from. The handle MUST be a file handle, and ACE4_READ_DATA MUST have been included in the desired-access when the file was opened.

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

start-offset
The starting offset of the data to hash.

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

quick-check-hash
The hash over the first 2048 bytes of the data range as the client knows it, or the entire range, if it is less than 2048 bytes. This allows the server to quickly check if it is worth the resources to hash a big file.

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

The response is either a SSH_FXP_STATUS packet, indicating an error, or the following extended reply packet:

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

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

9.2 Querying Available Space

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

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

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

```
byte   SSH_FXP_EXTENDED_REPLY
uint32 request-id
uint64 total-space-on-device
uint64 unused-on-device
uint64 total-space-available-to-user
uint64 unused-space-available-to-user
```

total-space-on-device
   The total amount of storage space on the device which stores
   'path', both used and unused, or 0 if unknown.

unused-space-on-device
The total amount of unused storage availabe on the device which stores 'path', or 0 if unknown.

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

unused-space-on-device
The total amount of unused storage available to the authenticated user on the device which stores 'path', or 0 if unknown.

10. Implementation Considerations

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

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

When implemented over ssh, the best performance should be achieved when the data size matches the channels max packet, and the channel window is a multiple of the channel packet size.

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

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

11. Security Considerations

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

This protocol provides file system access to arbitrary files on the server (only constrained 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 the SSH User Authentication Protocol [6].

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 SFTP session. For example, the sftp server daemon may have files which it has opened for its own purposes, and the client must not be able to access these files by specifying an arbitrary file handle string.

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

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

Servers should take care that file-system quotas are respected for users. In addition, implementations should be aware that attacks may be possible, or facilitated, by filling a filesystem. For example, filling the filesystem where event logging and auditing occurs may,

at best, cause the system to crash, or at worst, allow the attacker to take untraceable actions in the future.

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

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

12. Changes from Previous Protocol Versions

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

12.1 Changes Between Versions 6 and 5

```
**********  DO NOT IMPLEMENT **********
**********  DO NOT IMPLEMENT **********
*****                               *****
*****      There will be more edits after IETF 61.  *****
*****                               *****
**********  DO NOT IMPLEMENT **********
**********  DO NOT IMPLEMENT **********
```

- Add ability to negotiate version when client supports discontigous ranges of protocol version.
- Add 'filename-charset' and the 'filename-translation-control' extensions to allow better support of servers that can't reliably translate to UTF-8.
- Add DIR_NOT_EMPTY, NOT_A_DIRECTORY, INVALID_FILENAME and LINK_LOOP error codes.
- Added space-available extension.
- Added NOFOLLOW flag to open flags.
- Added allocation-size, text-hint, link-count, mime-type, and untranslated-name fields to attrib structure. Add ATTR_FLAGS_TRANSLATION_ERR to the attrib-bits.

12.2 Changes Between Versions 5 and 4

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

- Add "supported" extension to communicate features supported.
- Clarify error handling when client requests unsupported feature. (For example, attempts to write an unsupported attribute.)
- Add attrib-bits field to the attribute structure, which specifies a number of boolean attributes related to files and directories, including advisory read-only and case-sensitivity bits.

- Clarify the actual bit values to be used for the permissions field (since posix doesn't define values) and correct the value of ATTR_PERMISSIONS flag.
- Some reordering of sections to attempt to get a better grouping of related functionality.
- Open request explicitly specifies the access desired for the file.

- Add support for explicitly requesting file locking.
- Add support for better control of the rename operation.
- Add SSH_FX_NO_SPACE_ON_FILESYSTEM, SSH_FX_QUOTA_EXCEEDED, and SSH_FX_UNKNOWN_PRINCIPLE error codes.
- Add support for error specific data. This is used by a new SSH_FX_UNKNOWN_PRINCIPLE error to communicate which principles are unknown.
- Add support for retrieving md5-hash of file contents.
- Update security section.

12.3 Changes Between Versions 4 and 3

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

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

12.4 Changes Between Versions 3 and 2

- The SSH_FXP_READLINK and SSH_FXP_SYMLINK messages were added.
- The SSH_FXP_EXTENDED and SSH_FXP_EXTENDED_REPLY messages were added.
- The SSH_FXP_STATUS message was changed to include fields 'error

message' and 'language tag'.

12.5 Changes Between Versions 2 and 1

- The SSH_FXP_RENAME message was added.

12.6 Changes Between Versions 1 and 0

- Implementation changes, no actual protocol changes.

13. Trademark Issues

"ssh" is a registered trademark of SSH Communications Security Corp in the United States and/or other countries.

14. References

14.1 Normative References

[1] Ylonen, T. and C. Lonvick, "SSH Protocol Architecture", draft-ietf-secsh-architecture-16 (work in progress), June 2004.

[2] Ylonen, T. and C. Lonvick, "SSH Transport Layer Protocol", draft-ietf-secsh-transport-18 (work in progress), June 2004.

[3] Ylonen, T., Kivinen, T., Rinne, T. and S. Lehtinen, "SSH Connection Protocol", draft-ietf-secsh-connect-19 (work in progress), June 2004.

[4] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M. and D. Noveck, "NFS version 4 Protocol", RFC 3010, December 2000.

[5] 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.

14.2 Informative References

[6] Ylonen, T. and C. Lonvick, "SSH Authentication Protocol", draft-ietf-secsh-userauth-21 (work in progress), June 2004.

[7] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[8] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[9] 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.

Author's Address

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

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

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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