Network Working Group   T. Ylonen
Internet-Draft   S. Lehtinen
Expires: April 1, 2002   SSH Communications Security Corp
                                                       October 2001

SSH File Transfer Protocol
draft-ietf-secsh-filexfer-02.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](link). Introduction

   This protocol provides secure file transfer (and more generally file
   system access) functionality over a reliable data stream, such as a
   channel in the SSH2 protocol [3].

   This protocol 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, and that
   the server has already authenticated the user at the client end, and
   that the identity of the client user is externally available to the
   server implementation.

   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.[3].

   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 [1] 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 [5] as a subsystem, as
described in section ``Starting a Shell or a Command''. The
subsystem name used with this protocol is "sftp".

[3](#). General Packet Format

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

```
     uint32            length
     byte              type
     byte[length - 1]  data payload
```

 That is, they are just data preceded by 32-bit length and 8-bit type
 fields. The `length' is the length of the data area, and does not
 include the `length' field itself. The format and interpretation of
 the data area depends on the packet type.

 All packet descriptions below only specify the packet type and the
 data that goes into the data field. Thus, they should be prefixed by
 the `length' and `type' fields.

 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.

 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.

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
```

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 vendor-specific extensions. See
Section ``Vendor-Specific-Extensions'' for more details.

4. Protocol Initialization

When the file transfer protocol starts, it 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 SSH_FXP_INIT packet (from client to server) has the following
data:

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

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

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

The version number of the protocol specified in this document is 3.
The version number should be incremented for each incompatible
revision of this protocol.

The extension data in the above packets 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.

[5](#). **File Attributes**

   A new compound data type is defined for encoding file attributes. It
   is basically just a combination of elementary types, but is defined
   once because of the non-trivial description of the fields and to
   ensure maintainability.

   The same encoding is used both when returning file attributes from
   the server and when sending file attributes to the server. When
   sending it to the server, the flags field specifies which attributes
   are included, and the server will use default values for the
   remaining attributes (or will not modify the values of remaining
   attributes). When receiving attributes from the server, the flags
   specify which attributes are included in the returned data. The
   server normally returns all attributes it knows about.

```
      uint32   flags
      uint64   size          present only if flag SSH_FILEXFER_ATTR_SIZE
      uint32   uid           present only if flag SSH_FILEXFER_ATTR_UIDGID
      uint32   gid           present only if flag SSH_FILEXFER_ATTR_UIDGID
      uint32   permissions   present only if flag
SSH_FILEXFER_ATTR_PERMISSIONS
      uint32   atime         present only if flag SSH_FILEXFER_ACMODTIME
      uint32   mtime         present only if flag SSH_FILEXFER_ACMODTIME
      uint32   extended_count present only if flag SSH_FILEXFER_ATTR_EXTENDED
      string   extended_type
      string   extended_data
      ...      more extended data (extended_type - extended_data pairs),
                 so that number of pairs equals extended_count
```

   The `flags' specify which of the fields are present. Those fields
   for which the corresponding flag is not set are not present (not
   included in the packet). New flags can only be added by incrementing
   the protocol version number (or by using the extension mechanism
   described below).

   The `size' field specifies the size of the file in bytes.

   The `uid' and `gid' fields contain numeric Unix-like user and group
   identifiers, respectively.

   The `permissions' field contains a bit mask of file permissions as
   defined by posix [1](#).

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

   The SSH_FILEXFER_ATTR_EXTENDED flag provides a general extension

mechanism for vendor-specific extensions. 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.

 The flags bits are defined to have the following values:

```
    #define SSH_FILEXFER_ATTR_SIZE          0x00000001
    #define SSH_FILEXFER_ATTR_UIDGID        0x00000002
    #define SSH_FILEXFER_ATTR_PERMISSIONS   0x00000004
    #define SSH_FILEXFER_ATTR_ACMODTIME     0x00000008
    #define SSH_FILEXFER_ATTR_EXTENDED      0x80000000
```

6. **Requests From the Client to the Server**

   Requests from the client to the server represent the various file
   system operations. Each request begins with an `id' field, which is
   a 32-bit identifier identifying the request (selected by the client).
   The same identifier will be returned in the response to the request.
   One possible implementation of it is a monotonically increasing
   request sequence number (modulo 2^32).

   Many operations in the protocol operate on open files. The
   SSH_FXP_OPEN request can return a file handle (which is an opaque
   variable-length string) which may be used to access the file later
   (e.g. in a read operation). The client MUST NOT send requests 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.

6.1 **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.

[6.2](#) 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 ".." as referring to
   the parent directory, and "." as referring to the current directory.
   If the server implementation limits access to certain parts of the
   file system, it must be extra careful in parsing file names when
   enforcing such restrictions. There have been numerous reported
   security bugs where a ".." in a path name has allowed access outside
   the intended area.

   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.

[6.3](#) Opening, Creating, and Closing Files

    Files are opened and created using the SSH_FXP_OPEN message, whose
   data part is as follows:

```
        uint32        id
        string        filename
        uint32        pflags
        ATTRS         attrs
```

   The `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 `pflags' field is a bitmask. The following bits have been
defined.

```
#define SSH_FXF_READ            0x00000001
#define SSH_FXF_WRITE           0x00000002
#define SSH_FXF_APPEND          0x00000004
#define SSH_FXF_CREAT           0x00000008
#define SSH_FXF_TRUNC           0x00000010
#define SSH_FXF_EXCL            0x00000020
```

These have the following meanings:

SSH_FXF_READ
   Open the file for reading.

SSH_FXF_WRITE
   Open the file for writing. If both this and SSH_FXF_READ are
   specified, the file is opened for both reading and writing.

SSH_FXF_APPEND
   Force all writes to append data at the end of the file.

SSH_FXF_CREAT
   If this flag is specified, then a new file will be created if one
   does not already exist (if O_TRUNC is specified, the new file will
   be truncated to zero length if it previously exists).

SSH_FXF_TRUNC
   Forces an existing file with the same name to be truncated to zero
   length when creating a file by specifying SSH_FXF_CREAT.
   SSH_FXF_CREAT MUST also be specified if this flag is used.

SSH_FXF_EXCL
   Causes the request to fail if the named file already exists.
   SSH_FXF_CREAT MUST also be specified if this flag is used.

The `attrs' field specifies the initial attributes for the file.
Default values will be used for those attributes that are not
specified. See Section ``File Attributes'' for more information.

Regardless the server operating system, the file will always be
opened in "binary" mode (i.e., no translations between different
character sets and newline encodings).

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

A file is closed by using the SSH_FXP_CLOSE request. Its data field
has the following format:

        uint32   id
        string   handle

where `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. One
should note that on some server platforms even a close can fail.
This can happen e.g. if the server operating system caches writes,
and an error occurs while flushing cached writes during the close.

6.4 Reading and Writing

Once a file has been opened, it can be read using the SSH_FXP_READ
message, which has the following format:

        uint32   id
        string   handle
        uint64   offset
        uint32   len

where `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 `len'
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 `len'), 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 guaranteed that this will read the specified number of
bytes, or up to end of file. For e.g. device files this may return
fewer bytes than requested.

Writing to a file is achieved using the SSH_FXP_WRITE message, which
has the following format:

        uint32   id
        string   handle
        uint64   offset
        string   data

where `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 way 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 leave "holes" in the file.

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

6.5 Removing and Renaming Files

 Files can be removed using the SSH_FXP_REMOVE message. It has the
following format:

```
    uint32     id
    string     filename
```

where `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. Its data is as follows:

```
    uint32     id
    string     oldpath
    string     newpath
```

where `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. It is an error if there already exists a file
with the name specified by newpath. The server may also fail rename
requests in other situations, for example if `oldpath' and `newpath'
point to different file systems on the server.

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

6.6 Creating and Deleting Directories

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

```
uint32     id
string     path
ATTRS      attrs
```

where `id' is the request identifier, `path' and `attrs' specifies
the modifications to be made to its attributes. See Section ``File
Names'' for more information on file names. Attributes are discussed
in more detail in Section ``File Attributes''. specifies the
directory to be created. An error will be returned if a file or
directory with the specified path already exists. The server will
respond to this request with a SSH_FXP_STATUS message.

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

```
uint32     id
string     path
```

where `id' is the request identifier, and `path' specifies the
directory to be removed. See Section ``File Names'' for more
information on file names. An error will be returned if no directory
with the specified path exists, or if the specified directory is not
empty, or if the path specified a file system object other than a
directory. The server responds to this request with a SSH_FXP_STATUS
message.

6.7 Scanning Directories

The files in a directory can be listed using the SSH_FXP_OPENDIR and
SSH_FXP_READDIR requests. Each SSH_FXP_READDIR request returns one
or more file names with full file attributes for each file. The
client should call SSH_FXP_READDIR repeatedly until it has found the
file it is looking for or until the server responds with a
SSH_FXP_STATUS message indicating an error (normally SSH_FX_EOF if
there are no more files in the directory). The client should then
close the handle using the SSH_FXP_CLOSE request.

The SSH_FXP_OPENDIR opens a directory for reading. It has the
following format:

        uint32   id
        string   path

where `id' is the request identifier and `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. This will return
an error if the path does not specify a directory or if the directory
is not readable. The server will respond to this request with either
a SSH_FXP_HANDLE or a SSH_FXP_STATUS message.

 Once the directory has been successfully opened, files (and
directories) contained in it can be listed using SSH_FXP_READDIR
requests. These are of the format

        uint32   id
        string   handle

where `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.

When the client no longer wishes to read more names from the
directory, it SHOULD call SSH_FXP_CLOSE for the handle. The handle
should be closed regardless of whether an error has occurred or not.

6.8 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:

        uint32   id
        string   path

where `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.

    SSH_FXP_FSTAT differs from the others in that it returns status
   information for an open file (identified by the file handle). Its
   format is as follows:

```
    uint32     id
    string     handle
```

   where `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.

6.9 Setting File Attributes

   File attributes may be modified using the SSH_FXP_SETSTAT and
   SSH_FXP_FSETSTAT requests. These requests are used for operations
   such as changing the ownership, permissions or access times, as well
   as for truncating a file.

    The SSH_FXP_SETSTAT request is of the following format:

```
    uint32     id
    string     path
    ATTRS      attrs
```

   where `id' is the request identifier, `path' specifies the file
   system object (e.g. file or directory) whose attributes are to be
   modified, and `attrs' specifies the modifications to be made to its
   attributes. Attributes are discussed in more detail in Section
   ``File Attributes''.

   An error will be returned if the specified file system object does
   not exist or the user does not have sufficient rights to modify the
   specified attributes. The server responds to this request with a
   SSH_FXP_STATUS message.

    The SSH_FXP_FSETSTAT request modifies the attributes of a file which
   is already open. It has the following format:

```
    uint32     id
    string     handle
    ATTRS      attrs
```

   where `id' is the request identifier, `handle' (MUST be returned by
   SSH_FXP_OPEN) identifies the file whose attributes are to be
   modified, and `attrs' specifies the modifications to be made to its

attributes. Attributes are discussed in more detail in Section
``File Attributes''. The server will respond to this request with
SSH_FXP_STATUS.

[6.10](#) Dealing with Symbolic links

The SSH_FXP_READLINK request may be used to read the target of a
symbolic link. It would have a data part as follows:

```
    uint32     id
    string     path
```

where `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 will create a symbolic link on the
server. It is of the following format

```
    uint32     id
    string     linkpath
    string     targetpath
```

where `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 indicating either success (SSH_FX_OK) or an error
condition.

[6.11](#) 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:

```
    uint32     id
    string     path
```

where `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 only one name and a dummy attributes
value. The name is the returned packet will be in canonical form.

If an error occurs, the server may also respond with SSH_FXP_STATUS.

7. **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, any of the responses may
   be returned (depending on the operation). If no data needs to be
   returned to the client, the SSH_FXP_STATUS response with SSH_FX_OK
   status is appropriate. Otherwise, the SSH_FXP_HANDLE message is used
   to return a file handle (for SSH_FXP_OPEN and SSH_FXP_OPENDIR
   requests), SSH_FXP_DATA is used to return data from SSH_FXP_READ,
   SSH_FXP_NAME is used to return one or more file names from a
   SSH_FXP_READDIR or SSH_FXP_REALPATH request, and SSH_FXP_ATTRS is
   used to return file attributes from SSH_FXP_STAT, SSH_FXP_LSTAT, and
   SSH_FXP_FSTAT requests.

   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:

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

   where `id' is the request identifier, and `error/status code'
   indicates the result of the requested operation. The value SSH_FX_OK
   indicates success, and all other values indicate failure.

 Currently, the following values are defined (other values may be
defined by future versions of this protocol):

```
    #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
```

SSH_FX_OK
   Indicates successful completion of the operation.

SSH_FX_EOF
   indicates end-of-file condition; for SSH_FX_READ it means that no
   more data is available in the file, and for SSH_FX_READDIR it
   indicates that no more files are contained in the directory.

SSH_FX_NO_SUCH_FILE
   is returned when a reference is made to a file which should exist
   but doesn't.

SSH_FX_PERMISSION_DENIED
   is returned when the authenticated user does not have sufficient
   permissions to perform the operation.

SSH_FX_FAILURE
   is a generic catch-all error message; it should be returned if an
   error occurs for which there is no more specific error code
   defined.

SSH_FX_BAD_MESSAGE
   may be returned if a badly formatted packet or protocol
   incompatibility is detected.

SSH_FX_NO_CONNECTION
   is a pseudo-error which indicates that the client has no
   connection to the server (it can only be generated locally by the
   client, and MUST NOT be returned by servers).

SSH_FX_CONNECTION_LOST
   is a pseudo-error which indicates that the connection to the
   server has been lost (it can only be generated locally by the
   client, and MUST NOT be returned by servers).

SSH_FX_OP_UNSUPPORTED
   indicates that an attempt was made to perform an operation which
   is not supported for the server (it 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).

The SSH_FXP_HANDLE response has the following format:

       uint32   id
       string   handle

where `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:

       uint32   id
       string   data

where `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:

       uint32   id
       uint32   count
       repeats count times:
               string   filename
               string   longname
               ATTRS   attrs

where `id' is the request identifier, `count' is the number of names
returned in this response, and the remaining fields repeat `count'
times (so that all three fields are first included for the first
file, then for the second file, etc). 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),
`longname' is an expanded format for the file name, similar to what
is returned by "ls -l" on Unix systems, and `attrs' is the attributes

of the file as described in Section ``File Attributes''.

The format of the `longname' field is unspecified by this protocol.
It MUST be suitable for use in the output of a directory listing
command (in fact, the recommended operation for a directory listing
command is to simply display this data). However, clients SHOULD NOT
attempt to parse the longname field for file attributes; they SHOULD
use the attrs field instead.

 The recommended format for the longname field is as follows:

     -rwxr-xr-x   1 mjos   staff   348911 Mar 25 14:29 t-filexfer
     1234567890 123 12345678 12345678 12345678 123456789012

Here, the first line is sample output, and the second field indicates
widths of the various fields. Fields are separated by spaces. The
first field lists file permissions for user, group, and others; the
second field is link count; the third field is the name of the user
who owns the file; the fourth field is the name of the group that
owns the file; the fifth field is the size of the file in bytes; the
sixth field (which actually may contain spaces, but is fixed to 12
characters) is the file modification time, and the seventh field is
the file name. Each field is specified to be a minimum of certain
number of character positions (indicated by the second line above),
but may also be longer if the data does not fit in the specified
length.

 The SSH_FXP_ATTRS response has the following format:

     uint32   id
     ATTRS   attrs

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

8. Vendor-Specific Extensions

   The SSH_FXP_EXTENDED request provides a generic extension mechanism
   for adding vendor-specific commands. The request has the following
   format:

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

   where `id' is the request identifier, and `extended-request' is a
   string of the format "name@domain", where domain is an internet
   domain name of the vendor defining the request. The rest of the
   request is completely vendor-specific, 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:

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

9. **Security Considerations**

   This protocol assumes that it is run over a secure channel and that
   the endpoints of the channel have been authenticated. Thus, this
   protocol assumes that it is externally protected from network-level
   attacks.

   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].

   Care must be taken in the server implementation to check the validity
   of received file handle strings. The server should not rely on them
   directly; it MUST check the validity of each handle before relying on
   it.

10. Changes from previous protocol versions

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

10.1 Changes between versions 3 and 2

o The SSH_FXP_READLINK and SSH_FXP_SYMLINK messages were added.

o The SSH_FXP_EXTENDED and SSH_FXP_EXTENDED_REPLY messages were added.

o The SSH_FXP_STATUS message was changed to include fields `error message' and `language tag'.

10.2 Changes between versions 2 and 1

o The SSH_FXP_RENAME message was added.

10.3 Changes between versions 1 and 0

o Implementation changes, no actual protocol changes.

11. Trademark Issues

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

References

    [1]   Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and
          P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January
          1999.

    [2]   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.

    [3]   Rinne, T., Ylonen, T., Kivinen, T., Saarinen, M. and S.
          Lehtinen, "SSH Protocol Architecture", draft-ietf-secsh-
          architecture-09 (work in progress), July 2001.

    [4]   Rinne, T., Ylonen, T., Kivinen, T., Saarinen, M. and S.
          Lehtinen, "SSH Protocol Transport Protocol", draft-ietf-secsh-
          architecture-09 (work in progress), July 2001.

    [5]   Rinne, T., Ylonen, T., Kivinen, T., Saarinen, M. and S.
          Lehtinen, "SSH Connection Protocol", draft-ietf-secsh-connect-11
          (work in progress), July 2001.

    [6]   Rinne, T., Ylonen, T., Kivinen, T., Saarinen, M. and S.
          Lehtinen, "SSH Authentication Protocol", draft-ietf-secsh-
          userauth-11 (work in progress), July 2001.

Authors' Addresses

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

    EMail: ylo@ssh.com

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

    EMail: sjl@ssh.com

Full Copyright Statement

Acknowledgement