in: Development

Virtual1's Hotline Server Protocol Guide

EDIT

watch 01:25
Jurassic World: Dominion Dominates Fandom Wikis - The Loop

Do you like this video?

Play Sound

Contents

•• Virtual1's Hotline Server Protocol Guide ••

Version 1.50b Last Update: 02/12/99

Download current versions off the VirtualFTP Hotline server, at virtualftp.neotek.net

This guide was generated during several days of intense use of OTsessionWatcher, to get the protocol figured out in preparation for the development of HotSocket, a RealBasic socket-based class for use as a Hotline Client/Server interface.

Significant changes in 1.5:

- transaction 354 (userlist) is sent by server on login, the client does not need to request a userlist.

Thanks goes out to XAW and his development of the BHC, (Basic Hotline Client) whose RealBasic sourcecode gave me the insight necessary to begin to understand what I was seeing in the sessionwatcher. Thanks also to the creators of mBot, without whose greed and lack of interest in releasing sourcecode led me to make HotSocket, and thus, the need to make this guide. ;)

CHANGES HAVE BEEN MADE TO THE "PATH" TYPE. Please check the lengths of the bytes before the path strings, as they are handled differently than expected, and will not work the "old way" with 1.5's threaded news!

Chapter 1: NUMBERS AND STRINGS

long = 4 bytes short = 2 bytes

Anywhere there is a number that is preceeded by a length, (such as if the number is the only part of an object, like Socket or Icon) then the number can be a short OR a long. HL software will always pick the smaller of the two when sending, though it does not hurt them to receive a long that is zero.

Numbers appear to be stored as "two's complement".

```
0 = 00 00 00 00 (you can send as a short 00 00)
         +1 = 00 00 00 01 (you can send as a short
00 01)
     +65535 = 00 00 FF FF (you can send as a short
FF FF)
     +65536 = 00 01 00 00
+2 = 7F FF FF FF
−2147483648 = 80 00 00 00 (now counting backwards
toward zero)
         −1 = FF FF FF FF
```

This is how 2's complement works. The ctrl-F12 does not seem to parse this entirely correctly, but I am assuming this is how things are supposed to work internally in Hotline. The only place you'd have to worry about this is if you ran into a file > 2.1gb that was returning a negative filesize or something. Remember that icons can be negative numbers. (though the numbers are not likely to get near the "crossover" point at 231)

It might be simpler to just send everything you can as a long. Some

items must be sent as shorts if they don't have a length indicated in the protocol, such as all length indicators and some items in filelist/userlist entries. Anywhere you see short() or long(), it means that you MUST send it that way, because there is no length indicator. Anywhere you see number(), you need to send the length as a short, followed by the number, in your chosen format. There are a few oddball exceptions. Icon numbers are numbers, and normally the server will send them in Number format. (length followed by the number) Userlists however, send the Socket, Icon, and Status objects without length bytes, (all as shorts) Filelists have the same limitation. In the event of a negative icon number, (it can happen, and does work) the icon will be sent as a SHORT two's complement number. They are very easy to convert fortunately... just lop off the the first two characters of the number. -3 changes from FF FF FF FD to FF FD. This limits your numeric range to -32768 <-> +32767.

Strings are sent as a length (always a short) followed by the string's characters. Strings marked as "encoded" have each character of the string EOR'd with $FF. i.e. y=chr(255-asc(x)) It's not meant to be hard to crack, just hard to READ and easy to DO. Note that strings added to the protocol in HL1.5 are sent as "pascal" strings, and have a length specified by ONE BYTE, not two. (no consistency!) These are referred to as "pstring" instead of "string" for clarity below.

Chapter 2: OBJECTS

objects are sent under the following format:

```
• object header
  • short (object ID number)
  • short (object length)  does not count these
four header bytes
• object data
  if it's a number >=0 and <65536:
    • short (number)
  if it's a number >65535:
    • long (number)
  if it's a number <0:
    • long (232+number)
  if it is a string:
    • string  encoded strings have all chars EOF
$FF)
  if it is a filelistentry
    • file type  four characters, or "fldr" if
folder, or
                "alis" if unresolved alias
    • file creator  four characters, or long(0) if
folder
    • long (file size in bytes, zero if folder)
    • long (contained items, zero if app/doc)
    • long (filename length)
    • string (filename)
  if it is a Path
    • short (directory levels)
    • one or more directory levels
      • short (0)  not sure what it's for
      • byte (length of dir name)
      • string (dir name)
  if it is a userlistentry
    • short (socket)
    • short (icon)
    • short (status)
    • short (length of nick)
    • string (nick)
  if it is a datetime
    • short (base year – usually 1904)
    • short (0)
    • long (number of seconds this date is from
midnight, jan 1, base year)
```

```
  if it is a resumeinfo
    • "RFLT" – Resume FiLe Transfer
    • short (1)
    • 34 zeros = 8 x long(0)  +  1 x short(0)
    • short (2)
    • Data descriptor
      • "DATA"
      • long (index to start at)
      • long (0)
      • long (0)
    • Resource descriptor
      • "MACR"
      • long (index to start at)
      • long (0)
      • long (0)
  if it is a newsgroup
    • const ($33 31 31 33 – no idea why, but it's
always there)
    • long (post count)
    • pstring (category name)
    • const ($00)
    • posts
      • long (thread ID)
      • long (date)
      • long (parent thread ID)
      • const ($00 00 00 00)
      • short (message element count)
      • pstring (subject)
      • pstring (poster)
      • message elements
        • pstring (mime type)
        • short (post size?)
  if it's a newsfolderitem
    • byte (item type: $01=folder, $0A=category)
    • raw data (folder/category name, no length
byte)
```

Integer objects are preceeded by a length for a reason. Do not assume that just because the object you are expecting can only be a number 0-50 ,that it will have to be sent as a short. It could be sent as a long, and we don't want to break the socket for such a simple misunderstanding. The reverse is true for longs, they may be sending an icon number that is 5, and decide to save a few bytes and send it as a short. BEWARE.

client objects and their ID numbers:

```
ID#  Name           Object Type
100  errormsg       string
101  message        string
102  nick           string
103  socket         number
104  icon           number
105  login          encoded string  NOT encoded in
transaction #352
106  password       encoded string
107  xferID         number  the ID number of the
file transfer (usually 32 bit)
108  xfersize       number  size of file xfer, in
bytes (smaller for resumes!!)
109  parameter      number  specifies icon for
broadcast, also emote flag
110  privs          eight bytes  can make 64 flags,
only use 27
111  ???
112  status         number  0=black non-idle
113  ban            short (1)    include to make a
kick into a ban
114  chatwindow     four random bytes??  example:
84 47 5E 02
115  subject        string   the new subject of a
chat window

116 waiting count object
200  fileentry      filelistentry
201  filename       string
202  path           path
203  resumeinfo     resume
204  resumeflag     short (1)
205  infolongtype   string   "Text File"
206  infocreator    string   "Simpletext"
207  infosize       number
```

```
208  infocreated    datetime
209  infomodified   datetime
210  comment        string
211  newfilename    string
212  targetpath     path
213  infotype       string  the 4-char macos type
code (redundant, client already has C/T)
214  Quote          string
300  userlistentry  userlistentry
320  newsfolderitem newsfolderitem
321  catlist        newsgroup
322  category       string
325  newspath       path    this one revealed to me
its "true nature" in v1.5 !
326  threadID       number  the serial number of a
message, for threading
327  newstype       string
328  newssubject    string
329  author         string
330  newsdate       date
331  prevthread     number
332  nextthread     number
333  newsdata       string
334  unknown!       number?
```

Note! XferSize is the actual number of bytes in the file if it's a download, but it's the size of the file datablock (length of file - 146 - length of filename - length of comment) if it's an upload! :P (that's length PLUS 146 PLUS filename)

Chapter 3: TRANSACTIONS

Transaction are sent under the following format:

```
• header
  • short (transaction class)  0=info/request,
1=reply
  • short (transaction ID number)  server replies
are always zero
  • long (task number)
  • long (error code)  valid if this is a reply,
0=ok, 1=err
  • long (length of data block)
  • long (length of data block) yes, again.  I
don't know why.
• data
  • short (number of objects in transaction)
  • objects  can be one, many, or none
```

It would be wise to assume that objects can be passed in IN ANY ORDER. The other Hotline Client sockets I have seen thus far will crumble to dust if Hinks changes the order of the objects, and I just bet his clients and servers are designed to handle this. BEWARE.

Transaction IDs, classes, types, names, and objects:

```
ID# Cls Init    Type     Name
Object(s)
101  0  Client  request  GetNews             (no
objects passed)
  0  1  Server  reply    GetNews
message
102  0  Server  info     NewPost
message
103  0  Client  request  PostNews
message
104  0  Server  info     Broadcast
message
104  0  Server  info     Error
parameter,message
104  0  Server  info     PrivateMessage
socket,nick,message(,banflag)
105  0  Client  info     SendChat
message(,chatwindow)(,parameter)
106  0  Server  info     RelayChat
message(,chatwindow)
107  0  Client  request  Login
```

```
login,password,nick,icon
108  0  Client  request  SendPM
socket,message(,banflag)(,quote)
109  0  Server  info     Agreement
message
110  0  Client  request  Kick
socket(,ban)
111  0  Server  info     Disconnected
message
112  0  Client  request  CreatePchatWith
socket
  0  1  Server  reply    CreatePchatWith
chatwindow,socket,icon,status,nick
113  0  Server  info     InvitedToPchat
chatwindow,socket,nick
113  0  Client  info     AddToPchat
socket,chatwindow
114  0  Client  Info     RejectPchat
chatwindow
115  0  Client  request  RequestJoinPchat
chatwindow
  0  1  Server  reply    JoiningPchat
userlistentry(,userlistentry,...)(,subject)
116  0  Client  Info     LeavingPchat
chatwindow
117  0  Server  Info     JoinedPchat
chatwindow,socket,icon,status,nick
118  0  Server  Info     LeftPchat
chatwindow,socket
119  0  Server  Info     ChangedSubject
chatwindow,subject
120  0  Client  Request  RequestChangeSubject
chatwindow,subject
200  0  Client  request  FolderList
(path)
  0  1  Server  reply    FolderList
{fileentry}
201 (unused)
202  0  Client  request  Download
filename(,path)(,resumeinfo)
  0  1  Server  reply    Download
xfersize,xferID
203  0  Client  request  Upload
filename,xfersize(,path),(resumeflag)
  0  1  Server  reply    Upload
xferid(,resumeinfo)
204  0  Client  request  MoveToTrash
filename(,path)
205  0  Client  request  CreateFolder
filename(,path)
206  0  Client  request  GetFileInfo
filename(,path)
  0  1  Server  reply    GetFileInfo
infotype,infolongtype,infocreator,filename,

infocreated,infomodified,infosize(,comment)
207  0  Client  request  SetFileInfo
filename(,path),(newfilename OR comment)
208  0  Client  request  MoveFile
filename(,path)(,targetpath)
209  0  Client  request  MakeAlias
filename(,path)(,targetpath)
300  0  Client  request  GetUserList         (no
objects passed)
  0  0  Server  reply    GetUserList
{userlistentry}
301  0  Server  info     UserChange
socket,icon,nick(,status)
302  0  Server  info     UserLeave
socket
303  0  Client  request  GetUserInfo
socket
  0  1  Server  reply    GetUserInfo
message(,nick)
304  0  Client  info     ChangeNickIcon
icon,nick
350  0  Client  request  CreateUser
login,password,nick,privs
351  0  Client  request  DeleteUser           login
352  0  Client  request  OpenUser             login
(NOT ENCODED)
  0  1  Server  reply    OpenUser
login,password,privs(,nick)
353  0  Client  request  ModifyUser
nick,login,password,privs
354  0  Server  info     Userlist
{userlistentry}
355 broadcast
370  0  Client  request  NewsDirlist
(newsdir)
     1  Server  reply    NewsDirList
{newsfolderitem}
371  0  Client  request  NewsCatList
```

```
371  0  Client  request  NewsCatList
(newsdir)
     1  Server  reply    NewsCatList
(newsgroup)
380  0  Client  request  DeleteNewsDirCat
newspath (kills categories and dirs)
381  0  Client  request  MakeNewsDir
newspath,filename
382  0  Client  request  MakeCategory
newspath,category
400  0  Client  request  GetThread
newspath,threadid,newstype
     1  Server  reply    GetThread
newsdata,prevthread,nextthread,newssubject,
author,newstype,newsdate
410  0  Client  request  PostThread
newspath,threadid,newssubject,unknown
334,newstype,newsdata,
     1  Server  reply    PostThread
newspath,threadid,newssubject,unknown
334,newstype,newsdata,
411  0  Client  request  DeleteThread
newspath,threadid
```


Transactions dealing with files always include the filename. If the path is not included, root folder can be assumed. If the file is being moved or aliased, targetpath may also be included. If not, root is assumed as the target.

Transaction #105 (SendChat) is chat. When sent with a parameter of 1, it becomes an emote. Server reply to #352 always returns string(ctrl-G) as password. #353 must send a password string (chr(0)) if password was not changed. Returning string(ctrl-G) will result in that being the user's new password!

Unless otherwise specified, a successful task reply will have an error code of 0 and no objects. Unsuccessful tasks will reply with an error code of 1 and the errormsg object.

Server transaction #104 "Error" is used for when client sends a non-request that fails, such as trying to send public chat when they don't have chat privs. (probably a screw-up by Hinks, he should have made ALL transactions generate a reply, IMHO)

Note: reply to #303 (get info) will be missing the Nick object if you're getting info on a "ghost". (HotSocket will return "" - the HL client returns "Unnamed User") The HL client will not allow a user to set their name to blank. (spaces are OK tho)

Note: a Task is a reply to a request. The object(s) included in the Task are dependent on what the request was. The Task can be matched back to its request by using the task number portion of the header. It's probably possible to reuse task numbers, but don't re-issue a task number in a request until the current instance of that task number has been replied to! I have noticed that while the client can create tasks, the server cannot. This makes sense, because a server would eventually crash or eat up all available memory if it had to remember tasks until complete, assuming it was up a week or so and had clients dropping. (leaving tasks in the air)

Chapter 4: LOGGING IN

Before sending a login, you must establish a "pipe". Do this by connecting to the port and then exchanging this "handshake" with the server:

```
CLIENT HELLO
  • "TRTPHOTL"  identifies this is a hotline client
  • short (1) minimum server version this client is
compatible with?
  • short (2) client version?

TRTPHOTL (0) (1) (0) (2)

SERVER HELLO
  •"TRTP"
  • long (errorcode) – 0=OK you are connected,
1=rejected

TRTP (0) (0) (0) (0)
```

Once these have been exchanged, you can assume you are connected to a HL server and can proceed to login. Until you have received a success reply to your login transaction, the only other transaction you can submit is a request for disconnect. (I think all others are just ignored?)

Once logged in, you are by no means required to request a userlist, request news, or do anything else for that matter. Hinks' client will send the login and then immediately fire off a request for the agreement, userlist and news, before even receiving confirmation of a successful login. (how rude!)

Chapter 5: NOTES

I have seen many admins and co-admins running around kicking idle users, saying they are "taking up bandwidth". I was wondering if this was true, and did some pondering. A user that is completely idle (no file xfers) by themselves will take zero bandwidth. There WILL be some bandwidth needed though for each time a user in the userlist goes idle, goes active, changes nick or icon, leaves, arrives, or someone posts public chat. Each of these events requires a task to be sent to every user online, though the amount of data sent is quite small. (typically only 40 bytes or so) News posts also go to all users , (even w/o news privs!) and those can be relatively large in comparison to the other transactions. It sounds kind of silly, but it is in everyone's best interest that on busy file-serving server, you should be quiet and use chat only sparingly.

```
THE END     I hope this is useful for you!     :–)
– Virtual1
```

Categories