Star Trek: Voyager Elite Force Server Query Protocol

The protocol for server queries is largely identical to that of Quake 3 Arena. However, there are some minor changes. The game uses a master server to facilitate easier game play. It does not allow for match making.

In the following, server communication strings are shown that all start with four times 0xff, the character queue ÿÿÿÿ. If the reader gets this character queue wrong (actually looks like four y with dots on top), please don't get confused. They are four times the ASCII character no. 255 (all 8 bits set to 1) in a row, no gaps.

Announce Game Server to Masterserver (Heartbeat)

A game server may announce itself to the master server by simply sending the string ÿÿÿÿ`\heartbeat\`*`27960`*`\gamename\STEF1\`. The digit series *27960* corresponds to the port of the reporting server and can deviate accordingly. The master server will then attempt to query the server status *of the sender*. If this succeeds, the server will be added. The IP, the port and the protocol of the game server, as well as the information whether it is empty or full, are stored information. This means that the client is only informed about the servers it can play on. For Eliteforce 1.2 or newer the protocol number is 24 (or for older versions like 0.28 it can also be 22). On the other hand, the client already specifies whether it wants to receive full and/or empty servers when the master server queries it. If the request is filtered, it will be answered accordingly.

After that, no more things happen. There will be no response to the game server. The master also checks if the game server is still running. For this purpose there is a simple replacement mechanism: When a server is stopped (command `quit`) it sends a last heartbeat (actually two) and then goes offline. Since the master server does not receive an answer when querying the server status, it assumes that the server is dead and does not return it in client responses.

The game server, on the other hand, reports to the master server at regular intervals, like when the status changes from empty to not empty or when the map changes. This also has the effect that a message possibly not received by the master server will (hopefully) be received after some time, and the game server will then be listed.

The game server also reports for the transitions Empty → Not Empty and Full → Not Full, and vice versa. You can also send a heartbeat manually on the console (e.g. via rcon). So there are all in all eight situations in which a heartbeat is sent:

1. server goes online for the first time,
2. server goes offline,
3. server status changes from empty to occupied,
4. server status changes from occupied to empty,
5. server state changes from only occupied to full,
6. server state changes from full to only occupied,
7. console command `heartbeat` is issued and
8. server routinely gives a sign of life every five minutes (hard-coded by the *HEARTBEAT_MSEC*

variable of the server code in *sv_main.c*). This is the longest a running game server will run without sending a heartbeat, while actually set to announce itself.

This form also applies 1:1 to IPv6 master servers.

Withdraw Game Server from Master Server (Heartstop)

Apparently there is an additional mechanism in Eliteforce which is probably intended to explicitly withdraw from the master server. This is done with the following string: ÿÿÿÿheartstop*27960*\gamename\STEF1\. Also here the digit series *27960* is the port of the logging out game server. This logout is actually not necessary and deviates from the original Quake 3 implementation, see previous section Announce Game Server to Masterserver (Heartbeat).

This specific message is not sent by ioQuake3 based EF versions, so it does not exist for IPv6 connection.

Server List Query

The query of the game servers known to the master server by a client is initiated as follows: ÿÿÿÿgetservers *xx*, with *xx* as a numeric specification of the game protocol version. For Eliteforce since version 1.2 this is 24. Version 0.28 still had 22 as specification. Furthermore, the query can be extended by the extensions `full` (also returns full servers) and `empty` (also returns empty servers) (separated by space 0x20, ASCII character no. 32). A query that returns all Eliteforce 1.2 servers that the master server knows would look like this: ÿÿÿÿgetservers 24 empty full.

For IPv6 master servers this changes slightly to ÿÿÿÿgetserversExt EliteForce 24 ipv6 empty full. So an `Ext EliteForce` and (if wanted) `ipv6` is slipped in. Aside from that the rest is still the same. Also instead of a broadcast (only available for IPv4) the game sends its query to the multicast group `ioef`, or as actual IPv6 address `ff04::696f:6566`. If for some reason you feel the need to change this it is set by the config variable *net_mcast6addr*. Note: When `ipv6` is used, then *only* IPv6 host addresses are returned. If it is missing, both, IPv4 and IPv6 are sent back to the client.

Server List Response

The answer to the client then looks like this: ÿÿÿÿgetserversResponse *addressport1**addressport2*\…*addressportn*\EOT. The block ÿÿÿÿgetserversResponse and the block \EOT are mandatory. In between (if there are known servers) are the addresses and ports of the known servers. Each server block has the form *ipPort*. Here *ip* means eight bytes of the IP in hex representation. The first address block corresponds to the first hex pair. The address `192.168.0.1` becomes `c0a90001`. The block *Port* is specified directly after it. This is the server port as four bytes little endian unsigned short, or short, as you would write the hex number on paper. The default port 27960 then becomes 6d38, for example. Both values (IP blocks and port) are displayed with leading zeros to keep the number of characters at eight resp. four. The entire server would be represented as follows: `\c0a900016d38`. The total answer for the servers `192.168.0.1:27960`, `192.168.178.1:27961` and `123.34.56.78:9012` would look like this: ÿÿÿÿgetserversResponse \c0a900016d38\c0a9b2016d39\7b22384e2334\EOT. Please note

again that only those servers are displayed that have the same version as the client (EF 1.2 and newer: 24) and the empty or full ones only if the client explicitly requests this. Other servers are not reported back.

The above goes for Raven Software's master server, which might be considered the reference implementation. There are, however master server implementations out there, that do not abide the above outlines rule for the query response entirely. The master *efmaster.tjps.eu*, for example deviates in such a fashion that the hex values of game server IPs and ports are not lower case. Also the trailing \EOT is followed by three additional zero bytes (0x00).

For IPv6 this changes slightly to ÿÿÿÿgetserversExtResponse/<*IP and port*>\EOT (again with three trailing zero bytes). The IPv6 is a 128 bit(!) or 16 byte number (so no hex-number text), in the same order, as it would be noted down on a sheet of paper, but without the `:`. The port is also no longer noted as Hex digits, but directly as bytes, reducing them to two bytes in size. So essentially it is almost the same structure, just with an `Ext` slipped in and the IP + port as pure binary numbers. Such a reply will *also* still contain blocks, that would be returned for IPv4 only (*hex-IP and -port*). So a master server, that serves both protocols, IPv4 and IPv6, will return both types of hosts to an IPv6 server list request. The introducing \ resp. / determines, which kind of host is to be expected. This way one IPv6 list request to a dual stack master server will return all the hosts he knows.

Query a Game Server's Status

As described above, the master server queries the reporting game server at least once before adding it, instead of simply adding it untested. The corresponding query to the game server looks like this: ÿÿÿÿgetinfo xxx. Whereby xxx already works like it is written here. For a real status query, a challenge number (sequence of digits) may also be there. For the master server, however, this is irrelevant.

This looks the same for IPv6 queries.

Answer Game Server Status Query

The game server then replies with all kinds of information about itself. From the basic structure it looks like this:

ÿÿÿÿinfoResponse "*Values*"

The *Values* between the " characters usually contain more information than is necessary for the master server. Their structure is always *Identifier**Value*. The last identifier-value pair there is always \challenge*xxx*, the already mentioned challenge. With master server queries the value *xxx* is to be understood literally, with real status queries from clients it can look different. Already encountered values look like this:

- `game`: Which mod is played, e.g. standard `baseEF`, or also `pinball`.
- `voip`: Which voice over IP codec is used, e.g. `opus`.
- `g_needpass`: Does the server require a password for access? `1` = yes, `0` = no.
- `pure`: Value of the `sv_pure` variable on the server. If `1`, the client is only allowed to have the same `*.pk3` files as the server has.

- `gametype`: Which game type is set, e.g. 4 for CTF. Corresponds to the server variable `g_gametype`. See also Game Types.
- `sv_maxclients`: What is the maximum number of clients that can be connected? Digit sequence. Is used to determine the state `full` (full) for the query on the master server.
- `g_humanplayers`: How many human players are actually playing on the server? Digit sequence.
- `clients`: How many players are currently on the server? Also contains the bots, unlike `g_humanplayers`. Is used to determine the state `full` (full) and `empty` (empty).
- `mapname`: The technical(!) name of the map currently being played, e.g. `hm_voy1`.
- `hostname`: What name was given to the server? E.g. `EF Classic @ Mobile Infantrie`.
- `protocol`: Which version does the game server have? For EF from 1.2 on it is 24, but old versions also have 22 (version 0.28) or 23 (version 1.1).
- `gamename`: Which game is played? For EF this is „EliteForce“.

In the standard implementation, it is not necessary for the master server to test the game servers again before sending the client list. Since a logout does not necessarily have to arrive at the server (UDP), it may be useful to send a query to the servers at regular intervals and to sort out servers that are no longer accessible.

For IPv6 this result is the same.

Local Server Queries via Broadcast

In general, no master servers are required in the LAN, since the servers are simply detected via broadcast messages. For this purpose, the query described in section Query a Game Server's Status is sent to the broadcast address 255.255.255.255 instead of to a specific server. As a result, the request package arrives at all participants. If a server is in the same subnet, it will act as addressed and respond if an Internet or LAN server is running. All servers with ports between 27960 and 27963 are queried here as well.

If a game server is behind a firewall, NAT or desktop firewall that blocks either the game server port or broadcast requests, the server will not appear in the server browser. Servers accessible on ports other than 27960 to 27963 will also not be visible. Therefore it is recommended to stay in this range of ports when not using a master server.

For IPv6 the concept of broadcasts does not exist in the simplicity, as IPv4 knows it. Multicast addresses take their place. To query all servers in one's own network, the address will always be `ff04::696f:6566`. The prefix *ff04* marks the address as a multicast address. *696f:6566* marks the multicast group. When interpreted as ASCII characters, the multicast group turns into *ioef*.

Requesting Server Details

The above mentioned queries are all to be seen in the context of the identification of existing servers. However, the game servers are able to provide further information to the client, e.g. who is currently there or what map limits are set. The corresponding request looks like this: `ÿÿÿÿgetstatus`. The answer has the following form: `ÿÿÿÿstatusResponse`

\Description1\Value1\Description2\Value2…\Description\Value. The line break after Response is character 0x0a, or ASCII no. 10. The value pairs are very similar to those of a server status query. The list of available identifiers contains, among others, the following value pairs:

- protocol: 24, the network version of the server.
- difficulty: The level of difficulty of the bots.
- unlagged: The server has implemented the „unlagged“ mode. When this identifier appears, it is usually set to 1.
- sv_maxPing: Maximum ping allowed by players. If the value is 0, there is no limit.
- sv_minPing: Minimum allowed ping from players. If the value is 0, there is no limit.
- sv_dlRate: How many kB/sec. can be expected for the download from the server at most.
- sv_minRate: The minimum number of bytes/sec. used for the connection.
- sv_maxRate: The maximum number of bytes/sec. used for the connection.
- sv_hostname: The name set for the server, e.g. EF Classic @ Mobile Infantry.
- sv_dlURL: An extension of newer versions of the servers: On which address map or mod downloads can be done alternatively, if the client does not have a file. Not used by original EF.
- sv_allowDownload: Are map downloads allowed? 1 = yes, 0 = no.
- g_speed: What running speed is set? Default is 300.
- g_friendlyfire: Is friendly fire on? „1“ = yes, „0“ = no.
- sv_floodProtect: Has the server activated the flood protection? „1“ = yes, „0“ = no.
- sv_maxclients: The maximum number of players that can play on the server.
- fraglimit: How many kills of a player will end a DM round.
- timelimit: How many minutes does a map take to complete at most?
- capturelimit: After how many captures/points of a team a CTF map is terminated.
- dmflags: Flags that modify the game rules: no falling damage = 8, fixed field of vision = 16 and footsteps = 32, sums are allowed.
- g_maxGameClients: Unclear, from the name probably a limit, how many players are allowed to play on the server. But this is already done by sv_maxclients.
- g_gravity: How strong is gravity on the map? Default is 800.
- g_weaponrespawn: After how many seconds does a weapon reappear at a location after it was taken from there.
- version: Long variation of the server version, e.g. Lilium Voyager HM 1.39_GIT_9fcb8a0-2018-03-06 linux-x86_64 Apr 30 2018.
- com_gamename: What game is played there? For EF: EliteForce.
- com_protocol: Which communication protocol is used by the server? E.g. 24 or 26.
- g_gametype: Which Mode is played, for example, 4.
- mapname: Which map is played (technical name), e.g. hm_noon.
- sv_privateClients: How many private client slots are set? Default is 0.
- bot_minplayers: To how many players per team does the server fill the teams with bots? Default is 0.
- betadate: Unclear. One value received is Feb 22 2004/16:52:14.
- gamename: Which Game is played? E.g. „PiNBALL 2.0.0“ or „baseEF“.
- g_pModDisintegration: Is disintegration mode activated? 1 = yes, 0 = no.
- g_pModElimination: Is the elimination mode activated? 1 = yes, 0 = no.
- g_needpass: Does the server require a password to join? 1 = yes, 0 = no.
- g_allowVote: Does the server allow voting? 1 = yes, 0 = no.

In addition, you can also add non-functional variables that are sent as well, e.g.:

- Friendly fire: A free text variable (without technical function) (text).
- Administrator: Contains the name of the admin, if set (text).

- `Email`: Mail address of the admin, if set (text).
- `url`: Web page to server, if set (text).
- `Location`: Location of the server, if set (text).
- `Clan`: Name of the clan.

At the end of the answer there may be, separated by a line break (0x0a), a list of current players on the server, in the following form: *`Number1 Number2`* `"Nickname"` + (0x0a). *`Number1`* is the number of questions, *`Number2`* should be the ping of the player, `"Nickname"` is the name chosen by the player. The line break at the end applies to every player, including the last one. The answer also ends with a line break.

The request and the answer to it are both the same for IPv6.

[Back to the games database] [Back to Star Trek: Voyager Elite Force]