The unofficial DEM format description

Uwe Girlich

uwe@half-empty.de

v1.0.9, 1/8/1999

This document describes the DEM file format. This file format is the result of "recording" a game in Quake. This documentation covers the Quake versions 0.91 through 1.09.

Table of Contents

1. Introduction

1.1. Recording and Playback

Recording a game in Quake is as easy as playing it: you need some console commands to do it well.

To create a single player DEM file start the game as usual and use the console command *record name level [cdtrack]*. This starts `level` with the currently selected skill and writes a record in `name.dem`. The recording will be written during all the play and this record file may grow unpredictable. Make sure that you have some MBytes free disk space. To stop this recording use *stop* or even quit the whole game (*quit*). To play it back, use the commands *playdemo name* or *timedemo name*.

To create a multi player DEM file start a "listen" server (recording from a dedicated server doesn't work) and use again the *record* command. This starts the selected level and the player at the server is alone in this level. Now all the other clients can connect to the server as usual and play what they like (deathmatch or team). The recording lasts until the player at the listen server uses the *disconnect*, *stop* or *quit* command. The recording is from the point of view of the player at the listen server (client 1). The playback works as in the single player case.

1.2. Versions

In this document I'll discuss the DEM format used by the various versions of Quake.

Table 1. Covered Quake versions

`Exe:`	version	platform	note
?	0.91	MS-DOS, SVGA, 8bpp	
?	0.92	MS-DOS, SVGA, 8bpp	
?	0.92	Linux, X11, 8bpp	
?	1.00	MS-DOS, SVGA, 8bpp	Shareware version
17:38:28 Jul 12 1996	1.01	MS-DOS, SVGA, 8bpp	CD retail version
22:32:43 Aug 4 1996	1.01	Linux, X11, 8bpp	
22:32:43 Aug 4 1996	1.01	Linux, X11, DGA, 8bpp	
15:21:16 Sep 13 1996	1.05beta	MS-DOS, SVGA, 8bpp	
02:59:04 Sep 30 1996	1.06	MS-DOS, SVGA, 8bpp	
16:49:53 Nov 22 1996	1.06	Linux, X11, 8bpp	
22:44:36 Jan 17 1997	1.01	Linux, X11, 16bpp	leaked source, patched
?	1.07	MS-DOS, SVGA, 8bpp	Mission Pack #1
?	1.07	Linux, SVGA, 8bpp	
18:28:16 Mar 11 1997	1.08	MS-DOS, SVGA, 8bpp	Mission Pack #2
? Mar 21 1997	1.09	Win32, DirectX, 8bpp	
15:28:15 Aug 7 1997	1.09	Linux, SVGA, 8bpp	

00:36:23 Oct 14 1997	(X11 Quake 1.00) 1.09	Linux, X11, 8bpp, 16bpp
? Nov 7 1997	1.09	Win32, Glide
14:37:37 Nov 13 1997	(Linux GL 0.97) 1.09	Linux, SVGA, Glide, 16bpp

There is a small change in the Quake DEM format from 1.06 to 1.07. It will be discussed in the section Section 7.16. A new message was introduced in 1.07 too (see section Section 7.35. In version 1.08 the *temp_entity* message changed a bit (see section Section 7.24). In version 1.09 the CD track reading changed totally (see section Section 6.1).

I actually check my documentation with Linux Registered Quake 1.09 (x/squake) and compare it rarely with MS-DOS Registered Quake 1.06.

2. Basics on the used client/server architecture

Unlike DOOM and similar games Quake uses a “server” process (or even computer) which “does” all the game play. The “clients” (at least one) send to the server all input events (keys, mouse etc.) and receive all necessary information to draw the current picture. This prevents Quake from inconsistencies and the network load increases linear with the clients and not quadratic.

The communication between server and clients is an asynchronous one. If you don’t press any key, your computer won’t send any packets to the server. But you receive from time to time (the network is unpredictable) a packet to describe the state of your client. It is obvious, that these packets must contain some time stamp information, the positions of all monsters in sight and some player state information like the current weapon, ammo etc.

And exactly this is the DEM file format: the recording of all packets from the server to that client, who recorded the game (the first client). I call these packets “blocks of messages” and the single information (time, position, ammo etc.) “message”. In the original QuakeC code are some comments referring to “commands” instead of “messages” but I won’t change all my documentation after all.

The (listen) server process does the actual recording (file write access). Since every client gets all information to write the demo himself it was a matter of time until someone wrote a client proxy to record demos at the client side. I know two of them:

dproxy

: written by Kekoa Proudfoot kekoa@graphics.stanford.edu (mailto:kekoa@graphics.stanford.edu).

FAQ-Proxy

: written by Juha Kujala jmkujala@cc.jyu.fi (mailto:jmkujala@cc.jyu.fi) and Ilkka Rajala r151925@proffa.cc.tut.fi (mailto:r151925@proffa.cc.tut.fi). For more information visit FAQ homepage at http://www.modeemi.cs.tut.fi/quake/.

3. Some remarks on the used demo format

3.1. Advertising

As the clever reader may know I'm the author of LMPC, the LMP/DMO/DEM/QWD Control Centre. With this tool you may

- "decompile" an existing DEM file to a simple text file and
- "compile" such a (modified) text file back to a binary DEM file.

With LMPC it is very easy to analyse a DEM file but you can change it as well and so create a DEM file of a Quake game you never played. The current version of LMPC can be found at my Demo Specs page (http://demospecs.half-empty.de).

3.2. Difference to DOOM

The recording of a DOOM game consists only of the player input. All the rest is random-number dependent but totally deterministic and will be recalculated during the playback.

If you change a single action in a LMP file all the rest is garbage because all monsters now behave totally different and sooner or later (sooner) you run into a wall. This can't happen in a DEM file. The full movement of all objects is stored in it.

This confronts us with new opportunities but also new problems.

3.3. Opportunities of the DEM format

With the *centerprint* message it is possible to include some *sub-titles* in a recording file to inform the watchers what will happen next.

The player coordinates and the camera positions may be different. This makes it possible to simulate the Duke Nukem 3D feature of stationary cameras. The client doesn't draw the entity with the "viewpoint". This is in general the player entity itself but this entity can be changed to anything else with the setview message. Another problem is the entity selection of the server, which sends to the client only the entities in sight (of the client). Therefore it is impossible to enlarge the distance between the camera and the recording player too much. They both have to be on the same side of a wall.

For people with too much spare-time Quake can replace a full 3D modelling system for cartoons or the like.

The demo file can contain console commands, which the client runs during replay. With this feature it is possible to write a screen shot after every time stamp in the demo file. This makes it very easy to create a MPEG movie out of a DEM file.

3.4. Problems of the DEM format

It is trivial to remove the “godmode ON” and other cheat messages from a recording. All the action doesn’t change at all. These messages are only text print commands and the client behaviour doesn’t depend on them.

Fortunately I found a redundancy in the DEM format, which allows to detect a “godmode” cheater: Every damage message contains the health and armor decrease value. The next status line description (it contains the health and armor values to be displayed) can so be checked.

4. Some general remarks on the recording structure

4.1. Entity

An entity is an object. This may be the whole level (described by a BSP file), the player (described by a MDL file), an explosion (described by a SPR file) or the like.

There are different kind of entities.

4.1.1. Static Entity

A static entity doesn’t interact with the rest of the game. These are flames (`progs/flame.mdl`) and the like. It will be created by the spawnstatic message. It will never be necessary to reference such an entity. They don’t get an entity number. The maximum number of static entities is 127.

4.1.2. Dynamic Entity

A dynamic entity is anything which changes its behaviour or its appearance. These are ammunition boxes, spinning armours, player models and the like. A dynamic entity will be created by the spawnbaseline message. The maximum number of dynamic entities is 449.

4.1.3. Temporary Entity

A temporary entity will be created by the temp_entity message. A temporary entity is a (as the name indicates) short time entity.

Quake uses these entities for hits on the wall (point-like entities) or for the Thunderbolt flash (line-like entities).

For more information on temporary entities look in section Section 7.24.

4.2. Life-cycles

The Quake objects pass different life phases. The following information is not DEM specific but it may be of general interest to understand the cooperation of all the messages.

4.2.1. Armor

- To enable the client to display an armor the serverinfo message asks for the "progs/armor.mdl" model file and the "items/armor1.wav" sound file.
- The armor starts its life with a spawnbaseline message during the initialise phase. The armor is now a dynamic entity and spins around.
- The corresponding updateentity message appears only, if the camera is near enough to see the armor.
- The player gets it in the play. This results in the sound message "items/armor1.wav" and a print message "you got armor" and the stufftext message "bf\n" to make a short flash.
- The updateentity message for the armor doesn't appear ever again: the player got it.
- From this moment the corresponding bit in the `items` variable in the clientdata message will be 1 and the `armorvalue` variable get its maximum (100/150/200). From the `items` bit follows the colour of the picture to be drawn in the lower left corner of the status line.
- Now the player may be hit by a grenade. The total damage value (damage=take+save) will be split in take (`health-=take`) and save (`armorvalue-=save`). The save amount depends on the armor type (none/green/yellow/red): `save=0.0/0.3/0.6/0.8*damage`. The damage message in the DEM file tells the reduction of the current armor. With the old clientdata value and the reduction it is easy to recompute the new clientdata `armor` value. Any difference betrays the cheating player.
- After some severe hits the `armorvalue` variable is 0 and the `items` bit falls back to 0 as well. There is no armor anymore.

4.2.2. (Multi) Player

The following describes the deathmatch DEM messages of the two players Alice and Bob. Alice records the game from her `-listen 3` server.

- The serverinfo message contains the "maxclients 3" command to show how many clients are (at most) in this recording.
- During the 1st initialisation phase there are 3 spawnbaseline messages to create the player models. In the 2nd initialisation phase player 0 gets its name (Alice), colour and frag count (0) . The other 2 players get an empty name string. In the 3rd phase Alice gets again her name and colour. All these phases are controlled by signonnum messages.
- The game starts. Alice (entity 1) is alone in the game and looks around.

- Bob connects to Alice's server and it appears entity 2 (Bob's player model) a transport end temporary entity and a print message ("Bob entered the game") to inform everyone. Then the player 1 (Bob) gets his updatename and updatecolors message.
- Alice doesn't hesitate and runs for him and shoots him with the Shotgun. During every shot the clientdata message reduces the ammo count, the angles[0] command shows the wobble of Alice's screen and the weaponframe command selects the corresponding weapon frames. There is a sound message to start the `weapons/guncock.wav` file. Entity 1 gets its attack_state command. Alice hits Bob and so appear many particle messages (blood). Every Shotgun shot contains 6 parts. This means the shot can create anything from 6 particles (full hit) and 0 temporary entities (type 2: wall hits) to 0 particles and 6 temporary entities. If there was at least one particle Bob creates a sound message to start `player/pain?.wav`.
- Alice kills Bob. This creates the sound message to start `player/death1.wav`. Then comes the updatefrags message to give Alice 1 frag and a print message to inform everyone "Bob chewed on Alice's boomstick". A new entity will be created on the fly with an updateentity message to display Bob's backpack.
- Bob is dead, his entity 2 model remains in the death frame.
- After some seconds he starts again by pressing `SPACE`. He reappears in a totally different part of the level. The dead body transforms from entity 2 to entity 4 (maxclients+1) and a temporary entity (transport end) informs about his return. He is out of sight from the point of Alice's view. This means there is no entity 2 updateentity message.
- Bob runs to Alice's room. He goes through a slipgate and appears with 4 temporary entities (type 11: transport end) and the entity 2 in her room.
- Bob shoots and kills Alice. The scenario is the same as above. Only the damage messages appear now too, because Alice was hit.
- Bob uses the say console command (*say this sucks*) and in the DEM file appears a print message "\001Bob: this sucks".
- Bob disconnects from Alice's server. This results in a print message "Bob left the game with 1 frags" and updatename and updatecolor messages to remove client 2 (or player 1). It is a bit strange but there are 2 updatefrags messages: player 1 gets first 0 frags (this I understand) and then again 1 frag (this I don't understand at all).
- Entity 2 represents now the dead player 1.
- Alice spins around (it is possible even if you are dead) and the two dead bodies from Bob are totally white because they represent player 1 and he got (as he left) the updatecolor message with the standard colours 0 and 0. She is alone, restarts again her play, goes to the level end slipgate and get the ranking screen (intermission message) with only one player (Alice). Then she stops the recording. The DEM file ends with a disconnect message.

4.2.3. Medikits, Chthon, etc.

May be included later, if someone volunteers. Reading the QuakeC source is much easier.

5. Quake font

A string may contain any 8 bit characters except '\377' and it ends with '\000'. The special characters '\n' and '\r' have their normal meaning.

The Quake font is an extended ASCII font (7 bit) which contains in the upper half a similar font but with a different colour.

If the first character of a string for the print message is '\001', the Quake client plays the intercom sound `misc/talk.wav` and prints all following characters of the string with the highest bit set. Bit 7 bit will be set with '\002' at the first position of the string as well but this does not play the intercom sound. The special characters '\n' and '\r' are not affected by the meta characters '\001' and '\002'.

I used a simple DEM file (http://demospecs.half-empty.de/misc/qfont.html) to print all 252 ASCII characters.

6. File structure

To describe the file structure, which is very complicated, I use C like program fragments and `struct` definitions. This simplifies my task a lot.

I invented all used names (messages, variables etc.) for myself, took them from the Quake binary, QuakeEd but almost all from the QuakeC source.

6.1. CD track

Beside all the beauty in DEM files, there is a real mess called CD track at the very beginning of a DEM file.

All DEM files should start with an ASCII string of the CD track number which was given to the `record` console command. The string should be terminated by '\n'.

There are two totally different variants, how Quake reads this first bit in: the old `fscanf`-variant and the new step-by-step variant introduced with version 1.09.

6.1.1. `fscanf`

Quake reads the CD track bit with something like

```
FILE *fp;

int cdtrack;

fscanf(fp,"%i\n",&cdtrack);
```

This makes it possible (read the fscanf(3) man page!) to terminate the CD track string with any (positive) number of any whitespaces (' ', \t,\r,\n). You shouldn't come across anything more complicated than "4" or "-1" (default CD track of the level).

Because of the possibility of multiple whitespaces at the end, Quake can't playback DEM files, where the first byte of the `blocksize` of the first block (see next section) can be interpreted as a whitespace. Quake wont playback such a file at all! This often happens with recordings of the levels `start` and `hip1m3`.

The `fscanf` approach made it possible to create DEM files without any CD track at all at the beginning. The `fscanf` call return 0 instead of 1 but who cares? The only thing to take into account is that the first byte of this DEM file have to be no ASCII number or whitespace or minus sign. So `fscanf` can't find the integer number and reads nothing. The track value is undefined.

6.1.2. step-by-step

Newer Quake versions try to overcome the multiple whitespace problem of the `fscanf` variant and read the track "by hand".

The CD track parsing code goes as follows:

```
FILE *fp;
int cdtrack=0;
int sign=0;
unsigned char number;

while((number = ReadByte(fp)) != '\n') {
  if (number == '-')
    sign=1;
  else
    track = track*10 + number - '0';
}
if (sign)
  track=-track;
```

This code is much better but it can't detect files without any CD track header (like `finesc5.dem` from the Eschaton movie). The other thing is that it interprets "1-\n" as -1 but the old `fscanf` variant will be totally confused.

6.1.3. General hints

My LMPC program tries to read any kind of DEM file, so you have the possibility to change and even remove a CD track. To be most compatibe, use only numbers and a minus sign at the beginning.

To find out the method, which your copy of Quake uses for CD track parsing use the Quake CD track parsing analyser (http://demospecs.half-empty.de/misc/cd.html). This is a multi-variant DEM file, which can be parsed by both variants but with different results.

If you didn't give a CD track number to the `record` console command the CD track string is "-1\n". This means almost all DEM files start with "-1\n". If you gave a CD track but it is not a number at all the string is "0\n".

All the rest of the DEM file consists of "blocks" of "messages".

6.2. Block of Messages

At first some QuakeEd-like coordinate `typedef`'s:

```
typedef float vec_t;

typedef vec_t vec3_t[3];
```

This is the block structure:

```
typedef struct {
                  long            blocksize;
                  vec3_t          angles;
                  char            messages[blocksize];
                } block_t;
```

A block of messages starts with a size. Then 3 angles follow which describe the camera viewing direction. All the rest of a block are bytes which form one or more messages.

The messages in a block are the same messages as in the server to client network protocol. The block header is different and the camera angles are missing in the network protocol. For more information on the network protocol look in The Unofficial Quake Specs at the official Quake-editing support site, `http://www.gamers.org/dEngine/quake/spec/`.

Please note the missing camera angles in the network protocol. In an actual game every Quake client "knows" its viewing direction for itself and gets from the server only the position.

6.3. Message

This is the message structure:

```
typedef struct {
                  unsigned char ID;
                  char          messagecontent[???];
                } message_t;
```

The length of a message depends on its type (or `ID`).

6.4. Auxiliary routines

Here comes the definition of some small auxiliary routines to simplify the main message description. `get_next_unsigned_char`, `get_next_signed_char`, `get_next_short` and `get_next_long` are basic functions and they do exactly what they are called. Please note: `byte`, `char` or `short` will be converted to `long`. Second note: all multi-byte structures in the DEM file are Intel ordered.

In the following I often use a count variable

```
int i;
```

without declaration. I hope this does not confuses you.

```
long ReadByte
{
  return (long) get_next_unsigned_char;
}



long ReadChar
{
  return (long) get_next_signed_char;
}



long ReadShort
{
  return (long) get_next_short;
}



long ReadLong
{
  return get_next_long;
}
```

Note: A signed angle in a single byte. There are only 256 possible direction to look into.

```
vec_t ReadAngle
{
  return (vec_t) ReadChar / 256.0 * 360.0;
}



vec_t ReadCoord
{
  return (vec_t) ReadShort * 0.125;
}
```

The string reading stops at '\0' or after 0x7FF bytes. The internal buffer has only 0x800 bytes available.

```
char* ReadString
{
  char* string_pointer;
  char string_buffer[0x800];

  string_pointer=string_buffer;
```

```
  for (i=0 ; i<0x7FF ; i++, string_pointer++) {
    if (! (*string_pointer = ReadChar) ) break;
  }
  *string_pointer = '\0';
  return strdup(string_buffer);
}
```

7. List of all message types

The easiest way to explain a message is to give a short C like program fragment to parse such a message. It is not really the same code base as in LMPC but it should be *very* similar. Each message can be described by its `ID` or its name.

7.1. bad

`ID`

`0x00`

purpose

Something is bad. This message should never appear.

parse routine

none

7.2. nop

`ID`

`0x01`

purpose

No operation.

parse routine

none

7.3. disconnect

`ID`

`0x02`

purpose

Disconnect from the server. Stops the game.

parse routine

none

7.4. updatestat

`ID`

`0x03`

purpose

Updates directly any values in the player state.

variables

`long index;`

is the index in the `playerstate` array.

Table 2. `updatestat` indices

index	variable
0	`health`
1	??? (not used)
2	`weaponmodel`
3	`currentammo`
4	`armorvalue`
5	`weaponframe`
6	`ammo_shells`
7	`ammo_nails`
8	`ammo_rockets`
9	`ammo_cells`
10	`weapon`

11	`total_secrets`
12	`total_monsters`
13	`found_secrets`
14	`killed_monsters`
15	???
.	
.	
.	
31	???

Normal DEM files use index 11 to 14 only.

`long value;`

is the new value.

`long playerstate[32];`

is the internal array to describe the player state. Many other messages change (indirectly) some values in it.

parse routine

```
index = ReadByte;
if (index > 0x1F) {
  error("svc_updatestat: %i is invalid", index);
}
value = ReadLong;
playerstate[index] = value;
```

7.5. version

`ID`

`0x04`

purpose

This message defines the version of the server. I never found such a message in a DEM file. It may be absorbed by the `serverinfo` message.

variables

```
long serverprotocol;
```

is the version number of the server. It should be `0x0F` in Quake.

parse routine

```
serverprotocol = ReadLong;
if (serverprotocol != 0x0F) {
  error("CL_ParseServerMessage: Server is protocol %i instead of %i\n",
                                                    serverprotocol, 0x0F);
}
```

7.6. setview

`ID`

`0x05`

purpose

Sets the camera position to the origin of this entity.

variables

```
long entity;
```

is the entity with the camera.

parse routine

```
entity = ReadShort;
```

7.7. sound

`ID`

`0x06`

purpose

This message starts the play of a sound at a specific point.

variables

`long mask;`

is a bitmask to reduce the amount of data.

`float vol;`

is the volume of the sound (0.0 off, 1.0 max)

`float attenuation;`

is the attenuation of the sound.

Table 3. Sound attenuations

value	QuakeC	purpose
0	ATTN_NONE	i. e. player's death sound doesn't get an attenuation
1	ATTN_NORM	the normal attenuation
2	ATTN_IDLE	for idle monsters
3	ATTN_STATIC	for spawnstaticsound messages

`long channel;`

is the sound channel. There are 8 possible sound channels for each entity in Quake but the game uses 5 only.

Table 4. Sound channels

value	QuakeC	purpose
0	CHAN_AUTO	selects a channel automatically
1	CHAN_WEAPON	weapon use sounds
2	CHAN_VOICE	pain calls
3	CHAN_ITEM	item get sounds
4	CHAN_BODY	jump and fall sounds

`long entity;`

is the entity which caused the sound.

`long soundnum;`

is the sound number in the sound-table.

`vec3_t origin;`

is the origin of the sound.

parse routine

```
long entity_channel; // combined variable

mask = ReadByte;
vol = mask & 0x01 ? (float) ReadByte / 255.0 : 1.0;
attenuation = mask & 0x02 ? (float) ReadByte / 64.0 : 1.0;
entity_channel = ReadShort;
channel = entity_channel & 0x07;
entity = (entity_channel >> 3) & 0x1FFF;
soundnum = ReadByte;
for (i=0 ; i<3 ; i++) origin[i] = ReadCoord;
```

7.8. time

`ID`

`0x07`

purpose

This is the time stamp of a block of messages. A time message should appear in every game block.

variables

`float time;`

is the time in seconds from the beginning of the current level (not of the recording).

parse routine

```
time = ReadFloat;
```

7.9. print

ID

```
0x08
```

purpose

The client prints the text in the top left corner of the screen. There is space for 4 lines. They scroll up and the text disappears. The text will be printed on the console as well.

variables

```
char* text;
```

is the text to be displayed. The text contains something like “You get 5 shells”.

All font specials are explained in section Section 5.

parse routine

```
text = ReadString;
```

7.10. stufftext

ID

```
0x09
```

purpose

The client transfers the text to the console and runs it.

variables

```
char* text;
```

is the command, which the client has to execute. These are commands like “bf\n” to make a flash after you took something.

parse routine

```
text = ReadString;
```

7.11. setangle

`ID`

`0x0A`

purpose

This message set the camera orientation.

variables

`vec3_t angles;`

is the new camera orientation.

parse routine

```
for (i=0 ; i<3 ; i++) angles[i] = ReadAngle;
```

7.12. serverinfo

`ID`

`0x0B`

purpose

This message is usually the first messages in a DEM file and after a level change. It loads model and sound files.

variables

`long serverversion;`

is the protocol version of the server. Quake uses the version value 15 and it is not likely, that this will change.

`long maxclients;`

is the maximum number of clients in this recording. It is 1 in single player recordings or the number after the `-listen` command line parameter.

`long multi;`

is 0 in single player recordings and 1 in multi player recordings. It actually toggles the ranking screen at the end of a level.

`char* mapname;`

is the name of the level.

`char* precache_models[MAX_MODELS+1];`

is the model-table. It will be filled up with model names. Many other messages contain an index in this array. The first used index is 1. `MAX_MODELS` has the value 255.

`long nummodels;`

is the number of models in the model-table.

`char* precache_sounds[MAX_MODELS+1];`

is the sound-table. It will be filled up with sound names. Many other messages contain an index in this array. The first used index is 1. `MAX_SOUNDS` has the value 255.

`long numsounds;`

is the number of sounds in the sound-table.

parse routine

```
serverversion = ReadLong;
if (serverversion != PROTOCOL_VERSION) {
  error("Server returned version %i, not %i", serverversion, PROTOCOL_VERSION);
}
maxclients = ReadByte;
multi = ReadByte;
mapname = ReadString;
nummodels = 0;
do {
  if (++nummodels > MAX_MODELS) error("Server sent too many model_precache");
  precache_models[nummodels] = ReadString;
} while (*precache_models[nummodels]);
numsounds = 0;
do {
  if (++numsounds > MAX_SOUNDS) error("Server sent too many sound_precache");
  precache_sounds[numsounds] = ReadString;
} while (*precache_sounds[numsounds]);
```

7.13. lightstyle

`ID`

`0x0C`

purpose

This message defines a light style.

variables

`long style;`

is the light style number.

`char* string;`

is a string of letters “a” .. “z”, where “a” means black and “z” white. All effects from nervous flashing: “az” to slow dimming: “zyxwvu ... edcba” can so be described.

parse routine

```
style = ReadByte;
string = ReadString;
```

7.14. updatename

`ID`

`0x0D`

purpose

This message sets the player name.

`variables`

`long player;`

is the internal player number (client 1 has the player entity 0).

`char* netname;`

is the new player name.

parse routine

```
player = ReadByte;
netname = ReadString;
```

7.15. updatefrags

`ID`

`0x0E`

purpose

This message updates the frag count of a specific player.

variables

`long player;`

is the internal player number (client 1 has the player entity 0).

`long frags;`

is the new frag count for this player.

parse routine

```
player = ReadByte;
frags = ReadShort;
```

7.16. clientdata

`ID`

`0x0F`

purpose

This message updates the status line and the camera coordinates.

variables

`long mask;`

is a bitmask to show which values are coming.

`float view_ofs_z;`

is an additional viewing offset because the camera is at the origin of the entity and not at the eyes (is -8 if the player is death).

`float punchangle_x;`

is an additional offset of the first angle.

`vec3_t angles;`

indicates the camera direction change.

`vec3_t vel;`

indicates the camera velocity.

`long items;`

contains a bit mask for the inventory.

Table 5. `items` bits

bit	value	QuakeC	purpose
0	`0x00000001`	IT_SHOTGUN	Shotgun (should be always 1)
1	`0x00000002`	IT_SUPER_SHOTGUN	Double-barrelled Shotgun
2	`0x00000004`	IT_NAILGUN	Nailgun
3	`0x00000008`	IT_SUPER_NAILGUN	Perforator
4	`0x00000010`	IT_GRENADE_LAUNCHER	Grenade Launcher
5	`0x00000020`	IT_ROCKET_LAUNCHER	Rocket Launcher
6	`0x00000040`	IT_LIGHTNING	Thunderbolt
7	`0x00000080`	IT_EXTRA_WEAPON	extra weapon (there is no extra weapon)
8	`0x00000100`	IT_SHELLS	Shells are active
9	`0x00000200`	IT_NAILS	Nails are active
10	`0x00000400`	IT_ROCKETS	Grenades are active
11	`0x00000800`	IT_CELLS	Cells are active

12	`0x00001000`	IT_AXE	Axe (should be always 1)
13	`0x00002000`	IT_ARMOR1	green Armor
14	`0x00004000`	IT_ARMOR2	yellow Armor
15	`0x00008000`	IT_ARMOR3	red Armor
16	`0x00010000`	IT_SUPERHEALTH	Megahealth
17	`0x00020000`	IT_KEY1	silver keycard (or runekey or key)
18	`0x00040000`	IT_KEY2	gold keycard (or runekey or key)
19	`0x00080000`	IT_INVISIBILITY	Ring of Shadows
20	`0x00100000`	IT_INVULNERABILITY	Pentagram of Protection
21	`0x00200000`	IT_SUIT	Biosuit
22	`0x00400000`	IT_QUAD	Quad Damage
23	`0x00800000`	unknown	unknown (is 0)
24	`0x01000000`	unknown	unknown (is 0)
25	`0x02000000`	unknown	unknown (is 0)
26	`0x04000000`	unknown	unknown (is 0)
27	`0x08000000`	unknown	unknown (is 0)
28	`0x10000000`	unknown	Rune 1
29	`0x20000000`	unknown	Rune 2
30	`0x40000000`	unknown	Rune 3
31	`0x80000000`	unknown	Rune 4

You can find the default value for `items` in the parse routine: `0x4001`. It looks like a programmer's mistake because this means Shotgun any yellow Armor. It should be `0x1001`: Shotgun and Axe.

`long weaponframe;`

is the frame of the weapon model.

`long armorvalue;`

is the current armor.

`long weaponmodel;`

is the model number of the weapon in the model-table.

`long health;`

is the current health.

`long currentammo;`

is the current number of the current ammunition.

`long ammo_shells;`

is the current number of shells.

`long ammo_nails;`

is the current number of nails.

`long ammo_rockets;`

is the current number of rockets.

`long ammo_cells;`

is the current number of cells.

`long weapon;`

contains a bit mask for the current weapon.

Table 6. `weapon` bits

bit	value	QuakeC	weapon
?	`0x00`	not available	Axe
0	`0x01`	IT_SHOTGUN	Shotgun
1	`0x02`	IT_SUPER_SHOTGUN	Double-barrelled Shotgun
2	`0x04`	IT_NAILGUN	Nailgun
3	`0x08`	IT_SUPER_NAILGUN	Perforator
4	`0x10`	IT_GRENADE_LAUNCHER	Grenade Launcher
5	`0x20`	IT_ROCKET_LAUNCHER	Rocket Launcher
6	`0x40`	IT_LIGHTNING	Thunderbolt
7	`0x80`	IT_EXTRA_WEAPON	extra weapon (there is no extra weapon)

`float version;`

is the Quake version. Up to Quake 1.06 the bit `0x0200` in the `mask` variable indicated an used `items` entry. From 1.07 on the bit will be ignored and the `items` entry is compulsory. An old Quake client (<=1.06) can not play back the recording of a new Quake (>=1.07) because the unused bit is from 1.07 on always 0. The most compatible variant is to set the bit `0x0200` and include an items entry. This is the standard behaviour of LMPC. Even newer version of Quake (only checked with Linux Quake 1.09) set always the bit `0x0200` in the `mask` variable and send always the `items` entry.

parse routine

```
long uk_bit_b10, uk_bit_b11; // unknown

mask = ReadShort;
view_ofs_z = mask & 0x0001 ? (float) ReadChar : 22.0;
punchangle_x = mask & 0x0002 ? (float) ReadChar : 0.0;
angles[0] = mask & 0x0004 ? (vec_t) ReadChar : 0.0;
vel[0] = mask & 0x0020 ? (vec_t) ReadChar : 0.0;
angles[1] = mask & 0x0008 ? (vec_t) ReadChar : 0.0;
vel[1] = mask & 0x0040 ? (vec_t) ReadChar : 0.0;
angles[2] = mask & 0x0010 ? (vec_t) ReadChar : 0.0;
vel[2] = mask & 0x0080 ? (vec_t) ReadChar : 0.0;
if (version<=1.06)
  items = mask & 0x0200 ? ReadLong : 0x4001;
else
  items = ReadLong;
uk_bit_b10 = mask & 0x0400 ? 1 : 0; // bit 10
uk_bit_b11 = mask & 0x0800 ? 1 : 0; // bit 11
weaponframe = mask & 0x1000 ? ReadByte : 0;
armorvalue = mask & 0x2000 ? ReadByte : 0;
weaponmodel = mask & 0x4000 ? ReadByte : 0;
health = ReadShort;
currentammo = ReadByte;
ammo_shells = ReadByte;
ammo_nails = ReadByte;
ammo_rockets = ReadByte;
ammo_cells = ReadByte;
weapon = ReadByte;
```

7.17. stopsound

`ID`

`0x10`

purpose

Stops a sound. It looks for a sound started with a *sound* message with the same `channel` and `entity`.

variables

`long channel;`

is the sound channel.

`long entity;`

is the entity which caused the sound.

parse routine

```
long entity_channel; // combined variable

entity_channel = ReadShort;
channel = entity_channel & 0x07;
entity = (entity_channel >> 3) & 0x1FFF;
```

7.18. updatecolors

`ID`

`0x11`

purpose

Updates the colours of the specified player.

variables

`long player;`

is the internal player number (client 1 has player entity 0).

`long colors;`

is the combined colour of this player.

`long shirtcolor;`

is the colour of the shirt (0 <= `shirtcolor` <= 15).

`long pantscolor;`

is the colour of the pants (0 <= `pantscolor` <= 15).

parse routine

```
player = ReadByte;
colors = ReadByte;
shirtcolor = (colors>>4) & 0x0F;
pantscolor = colors & 0x0F;
```

7.19. particle

`ID`

`0x12`

purpose

This starts particles flying around. This happens, if a barrel explodes or blood particles fly after being hit by an axe, shells or nails.

variables

`vec3_t origin;`

is the origin of the particles.

`vec3_t vel;`

is the velocity of the particles.

`long color;`

is the colour of the particles (chunk 0, blood 73, barrel 75 and thunderbolt 225).

`long count;`

is the number of the particles.

parse routine

```
for (i=0 ; i<3 ; i++) origin[i] = ReadCoord;
for (i=0 ; i<3 ; i++) vel[i] = (vec_t) ReadChar * 0.0625;
count = ReadByte;
color = ReadByte;
```

7.20. damage

`ID`

`0x13`

purpose

Tells how severe was a hit and from which point it came.

variables

`long save;`

will be subtracted from the current armor.

`long take;`

will be subtracted from the current health.

`vec3_t origin;`

is the origin of the hit. It points to the weapon of a monster or player (not the origin of the monster entity) or it is (0,0,0) if the damage was caused by drowning or burning.

parse routine

```
save = ReadByte;
take = ReadByte;
for (i=0 ; i<3 ; i++) origin[i] = ReadCoord;
```

7.21. spawnstatic

`ID`

`0x14`

purpose

This message creates a static entity and sets the internal default values.

variables

`long StaticEntityCount;`

is the number of already started static entities. The maximum number is 127.

`long default_modelindex;`

is the model number in the model-table.

`long default_frame;`

is the frame number of the model.

`long default_colormap;`

is the colormap number to display the model.

`long default_skin;`

is the skin number of the model. This is used for things with different skins (like players or armors).

`vec3_t default_origin;`

is the origin of the entity.

`vec3_t default_angles;`

is the orientation of the entity.

`#define MAX_STATIC_ENTITIES 127`

is the maximum number of static entities in a given level.

parse routine

```
int data1, data2, data3;

if (StaticEntityCount > MAX_STATIC_ENTITIES) error("Too many static entities");
StaticEntityCount++;
default_modelindex = ReadByte;
default_frame = ReadByte;
default_colormap = ReadByte;
default_skin = ReadByte;
for (i=0 ; i<3 ; i++) {
  default_origin[i] = ReadCoord;
  default_angles[i] = ReadAngle;
}
```

7.22. spawnbinary

`ID`

`0x15`

purpose

This is OBSOLETE. It should never occur in DEM files.

parse routine

```
error ("CL_ParseServerMessage: Illegible server message\n");
```

7.23. spawnbaseline

`ID`

`0x16`

purpose

This message creates a dynamic entity and sets the internal default values.

variables

`long entity;`

is the number of the entity.

`long default_modelindex;`

is the model number in the model-table.

`long default_frame;`

is the frame number of the model.

`long default_colormap;`

is the colormap number to display the model.

`long default_skin;`

is the skin number of the model. This is used for things with different skins (like players or armors).

`vec3_t default_origin;`

is the origin of the entity.

`vec3_t default_angles;`

is the orientation of the entity.

```
#define MAX_ENTITIES 449
```

is the maximum number of entities in a given level.

parse routine

```
entity = ReadShort;
if (entity > MAX_ENTITIES) error("CL_EntityNum: %i is an invalid number", entity);
```

```
default_modelindex = ReadByte;
default_frame = ReadByte;
default_colormap = ReadByte;
default_skin = ReadByte;
for (i=0 ; i<3 ; i++) {
  default_origin[i] = ReadCoord;
  default_angles[i] = ReadAngle;
}
```

7.24. temp_entity

`ID`

`0x17`

purpose

This message creates a temporary entity.

variables

`long entitytype;`

is the type of the temporary entity. There are two kinds of temporary entities in Quake. `TE_EXPLOSION2` and `TE_BEAM` was introduced with the Mission Pack # 2 (Quake version 1.08).

point entity

is a small point like entity.

Table 7. point entities

value	QuakeC	purpose
0	TE_SPIKE	unknown
1	TE_SUPERSPIKE	superspike hits (spiketraps)
2	TE_GUNSHOT	hit on the wall (Axe, Shotgun)
3	TE_EXPLOSION	grenade/missile explosion
4	TE_TAREXPLOSION	explosion of atarbaby
7	TE_WIZSPIKE	wizard's hit
8	TE_KNIGHTSPIKE	hell knight's shothit
10	TE_LAVASPLASH	Chthon awakes and fallsdead

11	TE_TELEPORT	teleport end
12	TE_EXPLOSION2	other explosion

large entity

is a 2 dimensional entity.

Table 8. line entities

value	QuakeC	purpose
5	TE_LIGHTNING1	flash of theShambler
6	TE_LIGHTNING2	flash of theThunderbolt
9	TE_LIGHTNING3	flash in e1m7 to killChthon
13	TE_BEAM	grappling hook

`long entity;`

is the entity which created the temporary entity.

`vec3_t origin;`

is the origin of the entity.

`vec3_t trace_endpos;`

is the destination of the large temporary entity.

`long color;`

is the colour of the temporary entity.

`long range;`

is the range for a `TE_EXPLOSION2` explosion.

parse routine

```
entitytype = ReadByte;
switch (entitytype) {
  case TE_SPIKE:
  case TE_SUPERSPIKE:
  case TE_GUNSHOT:
  case TE_EXPLOSION:
  case TE_TAREXPLOSION:
  case TE_WIZSPIKE:
  case TE_KNIGHT_SPIKE:
  case TE_LAVASPLASH:
  case TE_TELEPORT:
```

```
        for (i=0 ; i<3 ; i++) origin[i] = ReadCoord;
  break;
  case TE_LIGHTNING1:
  case TE_LIGHTNING2:
  case TE_LIGHTNING3:
  case TE_BEAM:
        entity = ReadShort;
        for (i=0 ; i<3 ; i++) origin[i] = ReadCoord;
        for (i=0 ; i<3 ; i++) trace_endpos[i] = ReadCoord;
  break;
  case TE_EXPLOSION2:
        for (i=0 ; i<3 ; i++) origin[i] = ReadCoord;
        color = ReadByte;
        range = ReadByte;
  break;
  default:
    error("CL_ParseTEnt: bad type");
  break;
}
```

7.25. setpause

ID

0x18

purpose

Set the pause state. The time stands still but all entities get their update messages.

variables

```
long pausestate;
```

is 1 to start the pause and 0 to stop it.

parse routine

```
pausestate = ReadByte;
if (pausestate) {
  // pause is on
}
else {
  // pause is off
}
```

7.26. signonum

`ID`

`0x19`

purpose

This message selects the client state.

variables

long signon;

is the client state.

Table 9. `signon` values

value	purpose
1	after model/sound precache, start spawning entities (“prespawn”)
2	start initialising light effects
3	start 3D rendering

parse routine

```
signon = ReadByte;
```

7.27. centerprint

`ID`

`0x1A`

purpose

Prints the specified text at the centre of the screen. There is only one text line with a maximum of 40 characters. To print more than this one line, use ‘\n’ for a new line. Every text line (the first 40 characters) will be centred horizontally.

All font specials are explained in section Section 5.

variables

```
char* text;
```

is the text to be displayed.

parse routine

```
text = ReadString;
```

7.28. killedmonster

ID

```
0x1B
```

purpose

This message indicates the death of a monster.

variables

```
long killed_monsters;
```

is the number of killed monsters. It may be displayed with the console command *showscores*.

parse routine

```
killed_monsters++;
```

7.29. foundsecret

ID

```
0x1C
```

purpose

This message receives a client, if the player enters a secret area. It comes usually with a `print` message.

variables

`long found_secrets;`

is the number of found secrets. It may be displayed with the console command *showscores*.

parse routine

```
found_secrets++;
```

7.30. spawnstaticsound

`ID`

`0x1D`

purpose

This message starts a static (ambient) sound not connected to an entity but to a position.

variables

`vec3_t origin;`

is the origin of the sound.

`long soundnum;`

is the sound number in the sound-table.

`float vol;`

is the volume (0.0 off, 1.0 max)

`float attenuation;`

is the attenuation of the sound.

Table 10. Sound attenuation

value	QuakeC	purpose
0	ATTN_NONE	i. e. player's death sound doesn't get an attenuation
1	ATTN_NORM	the normal attenuation
2	ATTN_IDLE	attenuation for idle monsters

3	ATTN_STATIC	attenuation for spawnstaticsound messages

parse routine

```
for (i=0 ; i<3 ; i++) origin[i] = ReadCoord;
soundnum = ReadByte;
vol = (float) ReadByte / 255.0;
attenuation = (float) ReadByte / 64.0;
```

7.31. intermission

`ID`

`0x1E`

purpose

Displays the level end screen. Depending on the `multi` command in the serverinfo message this is either the single player summary screen or the multi player ranking screen.

parse routine

none

7.32. finale

`ID`

`0x1F`

purpose

Displays the episode end screen and some text.

variables

`char* text;`

is the episode end text.

parse routine

```
text = ReadString;
```

7.33. cdtrack

`ID`

`0x20`

purpose

This message selects the audio CD track numbers.

variables

`long fromtrack;`

is the start track.

`long totrack;`

is the last track. Both values are equal at the start of a game but become 2 and 3 at the end of an episode.

parse routine

```
fromtrack = ReadByte;
totrack = ReadByte;
```

7.34. sellscreen

`ID`

`0x21`

purpose

Displays the help and sell screen.

parse routine

none

7.35. cutscene

ID

`0x22`

purpose

This message appeared firstly with the Mission Pack #1 (Quake version 1.07). It is similar to *finale* as it displays an end screen and some text.

variables

```
char* text;
```

is the text.

parse routine

```
text = ReadString;
```

7.36. updateentity

ID

`>=0x80`

purpose

This is the general entity update message. For every entity (potentially) in sight the server sends such a message. The message contains only the values, which changed since the creation (or spawning) of the entity (with spawnstatic, spawnbaseline).

variables

`long mask;`

is a bit mask to reduce the amount of data to be sent. Only the changed parts (with respect to the initial state) get their bit and their values.

`long entity;`

is the entity number.

`long modelindex;`

is the model number in the model-table.

`long frame;`

is the frame number of the model.

`long colormap;`

is the colormap number to display the model.

`long skin;`

is the skin number of the model. This is used for things with different skins (like players or armors).

`long effects;`

contains a bit mask for special entity effects.

Table 11. Possible `effects` values

bit	value	QuakeC	purpose
0	`0x01`	EF_BRIGHTFIELD	not used
1	`0x02`	EF_MUZZLEFLASH	attack state of most entities
2	`0x04`	EF_BRIGHTLIGHT	not used
3	`0x08`	EF_DIMLIGHT	Quad Damage, Pentagram of Protection, Enforcer's laser

`vec3_t origin;`

is the origin of the entity.

`vec3_t angles;`

is the orientation of the entity.

`long new;`

is 1 if the entity gets some really new values (modelindex etc.)

parse routine

```
mask = ID & 0x07F;
if (mask & 0x0001) mask |= (ReadByte) << 8;
entity = mask & 0x4000 ? ReadShort : ReadByte;
modelindex = mask & 0x0400 ? ReadByte : default_modelindex;
frame = mask & 0x0040 ? ReadByte : default_frame;
colormap = mask & 0x0800 ? ReadByte : default_colormap;
skin = mask & 0x1000 ? ReadByte : default_skin;
effects = mask & 0x2000 ? ReadByte : default_effects;
origin[0] = mask & 0x0002 ? ReadCoord : default_origin[0];
angles[0] = mask & 0x0100 ? ReadAngle : default_angles[0];
origin[1] = mask & 0x0004 ? ReadCoord : default_origin[1];
angles[1] = mask & 0x0010 ? ReadAngle : default_angles[1];
origin[2] = mask & 0x0008 ? ReadCoord : default_origin[2];
angles[2] = mask & 0x0200 ? ReadAngle : default_angles[2];
new = mask & 0x0020 ? 1 : 0;
```

8. Version History and Acknowledgements

0.0.1, 7 July, 1996

- First version (working paper) completed.
- Many thanks to Steffen Winterfeldt (Steffen.Winterfeldt@itp.uni-leipzig.de (mailto:Steffen.Winterfeldt@itp.uni-leipzig.de)) for his unbelievable reverse engineering work. He worked out all the structure information.

0.0.2, 8 July, 1996

- Stupid spawnstatic error corrected.

0.0.3, 9 July, 1996

- I finally understood the multi player recordings.
- More info on sound, particle, spawnstaticsound.

0.0.4, 14 July, 1996

- Many new values decoded.
- Tables for weapons and status line.
- More general remarks.

0.0.5, 16 July, 1996

- Many new values decoded.
- Variables entry in the message description.
- Life-cycles.

1.0.0, 28 July, 1996

- QuakeC source is published. Many things get their right names now.
- Life-cycles for multi player.
- This version is the first reliable one.

1.0.1, 29 July, 1996

- Almost all identifier names match now the QuakeC names.
- Grammar check by SW.

1.0.2, 30 July, 1996

- Serious semantic mistake corrected (spawn/update).
- Some minor layout improvements.

1.0.3, 17 November, 1996

- I (finally) checked registered Quake: nothing special.
- effects and punchangle_x get their proper names.

1.0.4, 8 February, 1997

- Info on Quake font included.
- CD track header format finally corrected.

- Info on dproxy and FAQ-Proxy included.

1.0.5, 28 July, 1997

- Info on 1.06/1.07 problem included.
- Source is SGML-Tools 0.99.10 based.

1.0.6, 12 March, 1998

- SGML-Tools 1.0.5 used.
- First few Hexen II infos.
- New home is PlanetQuake.
- CD track section rewritten.
- Thanks to Stefan Schwoon (ssch0098@rz.uni-hildesheim.de (mailto:ssch0098@rz.uni-hildesheim.de)) for his hints on some 1.07 and 1.08 changes.

1.0.6, 12 March, 1998

- CD track section again beautyfied.

1.0.7, 15 July, 1998

- *stopsound* is OK now.
- SGML-Tools 1.0.7 used.

1.0.8, 6 September, 1998

- All Hexen II infos removed. It was never totally correct and I was never really interested in making it right.

1.0.9, 8 January, 1999

- typo corrected.