Quake MAP Specs

Last Updated: 8/22/96 8:52PM EST

Info on entities/maps and maintenance by Niklata (Nicholas Dwarkanath)
Additions and original HTMLization by Thomas Winzig

Here are the MAP file specs. This document should tell you everything you need to know about MAP files. This should help you in making editors/building MAP files. If you find an error, or figure something out that I don't know, please email Niklata. I'll put in your contribution and you'll get your name in here as well.

This document is now a part of the Unoffical Quake Specs. This standalone version is here to provide you with the most up-to-date information about Quake Map Editing. It is also a part of the Quake Documentation Project.

Since map editing is quite untested, a great deal of new information is pouring in about editing Quake levels. Not everything is here yet. I frequently update the seperate Map Specs and you can get the latest version from Niklata's web site. Please note that my web site will probably be moving soon! Keep checking for further information. I'll have it soon. Hopefully when editing becomes more mainstream and the rate of new discoveries comes to a near stop, I won't have to tell you about (or maintain) the standalone file.

You can get the most up-to-date version of the Quake MAP Specs from Quake Technical Resources.

Table of Contents

What's New

More bugfixes. Added stuff to coordinate systems and brushes.

Contributors to the Map Section

Thomas Winzig - Confirmation on the **model**; tag's behavior. HTMLization of original Map Specs doc. Clarified several things. Reorganized the document structure a little bit.
Brian Hackert - Info on how to use **func_train** and **path_corner**. Told me that **targetname** and **target** can have real

names for their argument, not just numbers.
Remco Stoffer - **trigger_counter** info, light styles, **trap_shooter**, some **trigger_relay** info, teleporting monsters, shootable buttons, and lots of other stuff.
Thomas Scherning - Information on the last five brush parameters.
Lars Bensmann - Information on using **origin** with attached brushes, a few bugfixes.
Robert Jones - Info on using **light** tag with fires, list of light styles.
Marc Fontaine - Correcting the Y-axis in the coord system.
Bernd Kreimeier - Correcting rot_angle and texture offsets.
Ben Morris - Correcting right/left handed system.

Dalias - Correcting several errors in section 2.2.3 and for func_wall's purpose.

Section 1: MAP Information

2.1.1 Coordinate System:

Quake uses a standard right-handed (X,Y,Z) coordinate system. If you're not familiar with the reference of "right-handed" to a coordinate system, it basically provides a tactile and visual discription of the mechanics of the system. If you point your fingers towards the positive x-axis and bend your fingers so that your knuckles face the positive y-axis, your thumb will point towards the positive z-axis:

```
       ^ z+
       |
       |
       |
       |
       |------------> y+
      /
     /
    /
   /
  <
x+
```

Some entities also need to have an **angle** tag that tells the direction it is facing. The values possible are listed below:

```
0-359: Normal Angle (zero == east)
-1: Up
-2: Down
```

2.1.2 Brushes:

Brushes are one of the two primary components of a MAP file. Each brush defines a solid region. Brushes define this region as the intersection of four or more planes. Each plane is defined by three noncolinear points. These points must go in a clockwise orientation:

```
1--2----------------->
|
3
|
|
|
|
|
,
```

Each brush statement looks like this:

```
 {
  ( 128 0 0 ) ( 128 1 0 ) ( 128 0 1 ) GROUND1_6 0 0 0 1.0 1.0
  ( 256 0 0 ) ( 256 0 1 ) ( 256 1 0 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 128 0 ) ( 0 128 1 ) ( 1 128 0 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 384 0 ) ( 1 384 0 ) ( 0 384 1 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 0 64 ) ( 1 0 64 ) ( 0 1 64 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 0 128 ) ( 0 1 128 ) ( 1 0 128 ) GROUND1_6 0 0 0 1.0 1.0
 }
```

That's probably just a bit confusing when you first see it. It defines a rectangular region that extends from (128,128,64) to (256,384,128). Here's what a single line means:

```
  ( 128 0 0 ) ( 128 1 0 ) ( 128 0 1 ) GROUND1_6 0 0 0 1.0 1.0
   1st Point   2nd Point   3rd Point   Texture
```

Now, you're probably wondering what those last five numbers are. I've listed what they do below:

```
x_off     - Texture x-offset (must be multiple of 16)
y_off     - Texture y-offset (must be multiple of 16)
rot_angle - floating point value indicating texture rotation
x_scale   - scales x-dimension of texture (negative value to flip)
y_scale   - scales y-dimension of texture (negative value to flip)
```

2.1.3 General MAP Info:

The actual MAP file format is quite simple. It is simply a text file. All Quake editing tools should support either UNIX or DOS text formats, as id's tools do.

MAP files are the development format for Quake levels. It is preferred that all editors work with MAP files, as all tools released by id Software also use the MAP file format. A map file is basically un-'compiled' level data. The actual MAP file is totally unusable by Quake itself until it is coverted to a BSP file by a BSP builder such as id Software's QBSP. For a MAP file's lightmaps to be calculated by Quake, the level must also be run through a light builder, such as LIGHT. Otherwise, everything will be displayed at full brightness. Finally, to speed up the level and ensure proper display, visibility lists should be calculated by a program such as VIS. The process for building levels in Quake is quite time-consuming compared to building Doom levels, even on a fast computer.

The generalized format for a MAP file is as follows:

```
{
 entity
 {
  brush (optional)
 }
}
...
```

Comments in the MAP file are indicated by a "//" (C++ style comment).

Many entity/brush combinations can be put into a map file. All MAP files must contain with a **worldspawn** entity, usually as the first entry. This entry defines all of the normal brushes that make up the structure of the level. There should be only one **worldspawn** entity per MAP file. Here's the syntax of the **worldspawn** class:

```
"classname" "worldspawn" // Tells Quake to spawn the world
"wad" "DIRPATH"          // tells what graphics (texture) WAD2 file to use.
"message" "TITLE"        // The title of the level
"worldtype" "#"          // Describes time of environment (changes appearance/name of keys)
                         // 0 == Medieval (medieval)
                         // 1 == Runic (metal)
                         // 2 == Present (base)
"sounds" "#"             // Tells the CD player which track to play.
"light" "#"              // Default light level
```

A simple map file would look like this:

```
{
 "sounds" "1"
 "classname" "worldspawn"
 "wad" "/gfx/base.wad"
 "worldtype" "0"
 {
  ( 128 0 0 ) ( 128 1 0 ) ( 128 0 1 ) GROUND1_6 0 0 0 1.0 1.0
  ( 256 0 0 ) ( 256 0 1 ) ( 256 1 0 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 128 0 ) ( 0 128 1 ) ( 1 128 0 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 384 0 ) ( 1 384 0 ) ( 0 384 1 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 0 64 ) ( 1 0 64 ) ( 0 1 64 ) GROUND1_6 0 0 0 1.0 1.0
  ( 0 0 128 ) ( 0 1 128 ) ( 1 0 128 ) GROUND1_6 0 0 0 1.0 1.0
 }
}
```

```
{
 "classname" "info_player_start"
 "origin" "256 384 160"
}
```

As you can see, all brushes are contained in entities, even those that make up the main level. The most complex part of MAP files are the entities. They are what the rest of this document are about.

Section 2: Entity information

2.2.1 Entity Overview:

Entities are the second major component of Quake MAP files. An entity is basically a bit like a thing, but they also function as triggers and as pathmarkers. A entity statement looks like this:

```
{
 "classname" "light"
 "origin" "0 128 64"
 "light" "255"
}
```

This is what is called a general entity statement. It is called a general statement because it does not attach to a brush. An attached entity statement looks like this:

```
{
 "classname" "func_door"
 "angle" "0"
 "speed" "16"
 "targetname" "t1"
 "sounds" "1"
 "wait" "16"
  {
   ( 128 0 0 ) ( 128 1 0 ) ( 128 0 1 ) GROUND1_6 0 0 0 1.0 1.0
   ( 256 0 0 ) ( 256 0 1 ) ( 256 1 0 ) GROUND1_6 0 0 0 1.0 1.0
   ( 0 128 0 ) ( 0 128 1 ) ( 1 128 0 ) GROUND1_6 0 0 0 1.0 1.0
   ( 0 384 0 ) ( 1 384 0 ) ( 0 384 1 ) GROUND1_6 0 0 0 1.0 1.0
   ( 0 0 64 ) ( 1 0 64 ) ( 0 1 64 ) GROUND1_6 0 0 0 1.0 1.0
   ( 0 0 128 ) ( 0 1 128 ) ( 1 0 128 ) GROUND1_6 0 0 0 1.0 1.0
  }
}
```

Attached entity brushes can have an "origin" tag. It can be used to offset where they appear in the level.

For the rest of the document, when I give you frameworks for a structure, the individual entries can be in any order, and lots are optional. I try to mark if an entry is optional, although this has not yet been rigorously tested.

In a "" block, your choices for that block are delimited by commas.

2.2.2 "spawnflags" tag:

I believe that anything with an origin tag can have an optional 'spawnflags' tag. This is not confirmed, however. Tags can be combined by addition or bitwise or (really the same thing).

```
General:

256  - Normal Skill or higher
512  - Hard Skill or higher
1792 - Appears in deathmatch only

item_health:
1    - Larger Health/Larger Ammo
2    - Megahealth

monster_zombie:
1    - Crucified Zombie
```

2.2.3 General Entity syntax:

(Thanks to John Wakelin who wrote much of this section)

The entities define the monsters, things, but also the positions in space where something must happen. So they are the Quake equivalent of both the THINGS and the LINEDEF types from DOOM.

The entity definitions are made up of a series of specific details that define what each is, where it starts, when it appears etc. Each specific is followed by a modifier that arguments it. All definitions have the `classname` specific that identifies that entity. The `classname` specifics relate intimately with the code lump and are the names of functions written in Quake C.

I have chosen the terms ``specific" and ``arg" for the two different parts of each detail of the definition. These terms may or may not suit but, they will have to do until we learn what id calls them.

```
Specifics    Args      Description
-------------------------------------------------------------------------------
"classname"  "name"    // Type of entity to be defined (mandatory)
"origin"     "X Y Z"   // Coordinates of where it starts in space.
"angle"      "#"       // Direction it faces or moves (sometimes in degrees)
"light"      "#"       // Used with the light classname.
"target"     "t#"      // Matches a targetname.
"targetname" "t#"      // Like a linedef tag.
"killtarget" "#"       // Removes target when triggered?
"spawnflags" "#"       // Used to flag/describe an entity that is not default.
"style"      "#"       // Used to flag/describe an entity that is not default.
"message"    "string"  // Message displayed when triggered (\n for linebreaks)
"mangle"     "X Y Z"   // Point where the intermission camera looks at

{BRUSH INFO}           // In entities that describe triggers/doors/platforms, etc,
                       // the brush info is inserted into the entity brackets,
                       // delimited by another set of brackets.

specifics/args present only in models:

"speed"      "#"       // How fast the model is moved.
"wait"       "#"       // How long a pause between completion of movement or
                       // return to original position (in seconds or 10ths)
"lip"        "#"       // Seems to be a means of adjusting the starting position.
"dmg"        "#"       // How much damage the model causes when it shuts on you?
"health"     "#"       // How much damage the model takes before it triggers
"delay"      "#"       // Time before event is triggered
"sounds"     "#"       // How much damage the model causes when it shuts on you?
"wad"        "wadfile" // The wad2 graphics file used by the world for textures.
"height"     "#"       // How high a platform will raise


--------------------------------------------------------------------------------
{
    "specific1" "arg1" // The first descriptors (usually classname)
    "specific2" "arg2" // The second
    ...                // Etc...

    {
        <INSERT BRUSH INFO HERE> // for entities like doors/triggers/platforms/etc
    }
}
```

Note: The term *model* refers to a combination of a brush and an entity. One or more brushes are bound to an entity, which controls the behavior of the brushes. All brushes are contained within models.

The model numbers in the compiled BSP (**x*) comes from the order in which the models are stored in the models structure. These numbers are originally derived from the order of the models in the source MAP file.

The worldspawn model is a bounding box that defines the extents of the whole world.

The models are defined by a bounding box of the `max` and `min(x,y,z)`. Therefore they are always parallel to the horizontal planes. This bounding box is simply used for speeding up collision detection and will not affect the movement of the models themselves.

2.2.4 All known entities (current for Shareware Quake 1.01):

air_bubbles	: Rising bubbles
ambient_drip	: Dripping sound
ambient_drone	: Engine/machinery sound
ambient_comp_hum	: Computer background sounds
ambient_flouro_buzz	: Flourescent buzzing sound
ambient_light_buzz	: Buzzing sound from light
ambient_suck_wind	: Wind sound
ambient_swamp1	: Frogs croaking
ambient_swamp2	: Slightly different sounding frogs croaking
ambient_thunder	: Thunder sound
event_lightning	: Lightning (Used to kill Cthon, shareware boss)
func_door	: Door
func_door_secret	: A door that is triggered to open
func_wall	: A moving wall?
func_button	: A button
func_train	: A platform (moves along a "train")
func_plat	: A lift/elevator
func_dm_only	: A teleporter that only appears in deathmatch
func_illusionary	: Creates brush that appears solid, but isn't.
info_null	: Used as a placeholder (removes itself)
info_notnull	: Used as a placeholder (does not remove itself)
info_intermission	: Cameras positioning for intermission (?)
info_player_start	: Main player starting point (only one allowed)
info_player_deathmatch	: A deathmatch start (more than one allowed)
info_player_coop	: A coop player start (more than one allowed)
info_player_start2	: Return point from episode
info_teleport_destination	: Gives coords for a teleport destination using a targetname

All **item_** tags may have a **target** tag. It triggers
the event when the item is picked up.

item_cells	: Ammo for the Thunderbolt
item_rockets	: Ammo for Rocket/Grenade Launcher
item_shells	: Ammo for both Shotgun and SuperShotgun
item_spikes	: Ammo for Perforator and Super Perforator
item_weapon	: Generic weapon class
item_health	: Medkit
item_artifact_envirosuit	: Environmental Protection Suit
item_artifact_super_damage	: Quad Damage
item_artifact_invulnerability	: Pentagram of Protection
item_artifact_invisibility	: Ring of Shadows (Invisibility)
item_armorInv	: Red armor
item_armor2	: Yellow armor
item_armor1	: Green armor
item_key1	: Silver Key
item_key2	: Gold Key
item_sigil	: Sigil (a rune)
light	: A projected light. No visible lightsource.
light_torch_small_walltorch	: Small wall torch (gives off light)
light_flame_large_yellow	: Large yellow fire (gives off light)
light_flame_small_yellow	: Small yellow fire (gives off light)
light_flame_small_white	: Small white fire (gives off light)
light_fluoro	: Fluorescent light? (Gives off light, humming sound?)
light_fluorospark	: Fluorescent light? (Gives off light, makes sparking sound)
light_globe	: Light that appears as a globe sprite
monster_army	: Grunt
monster_dog	: Attack dog
monster_ogre	: Ogre
monster_ogre_marksman	: Ogre (synonymous with monster_ogre)
monster_knight	: Knight
monster_zombie	: Zombie
monster_wizard	: Scragg (Wizard)
monster_demon1	: Fiend (Demon)
monster_shambler	: Shambler
monster_boss	: Cthon (Boss of Shareware Quake)
monster_enforcer	: Enforcer
monster_hell_knight	: Hell Knight
monster_shalrath	: Shalrath
monster_tarbaby	: Slime

monster_fish	: Fish
monster_oldone	: Shubb-Niggurath (requires a misc_teleportrain and a info_intermission)
misc_fireball	: Small fireball (gives off light, harms player)
misc_explobox	: Large Nuclear Container
misc_explobox2	: Small Nuclear Container
misc_teleporttrain	: Spiked ball needed to telefrag monster_oldone
path_corner	: Used to define path of **func_train** platforms
trap_spikeshooter	: Shoots spikes (nails)
trap_shooter	: Fires nails without needing to be triggered.
trigger_teleport	: Teleport (all trigger_ tags are triggered by walkover)
trigger_changelevel	: Changes to another level
trigger_setskill	: Changes skill level
trigger_counter	: Triggers action after it has been triggered **count** times.
trigger_once	: Triggers action only once
trigger_multiple	: Triggers action (can be retriggered)
trigger_onlyregistered	: Triggers only if game is registered (registered == 1)
trigger_secret	: Triggers action and awards secret credit.
trigger_monsterjump	: Causes triggering monster to jump in a direction
trigger_relay	: Allows delayed/multiple actions from one trigger
trigger_push	: Pushes a player in a direction (like a windtunnel)
trigger_hurt	: Hurts whatever touches the trigger
weapon_supershotgun	: Super Shotgun
weapon_nailgun	: Perforator
weapon_supernailgun	: Super Perforator
weapon_grenadelauncher	: Grenade Launcher
weapon_rocketlauncher	: Rocket Launcher
weapon_lightning	: Lightning Gun

Section 3: Entity 'class' examples

2.3.1 Lights

For all light-emmitting entities, **spawnflags** and **style** have special meanings:

```
Spawnflags:
0 - Light starts on.  Switches off when triggered.
1 - Light starts off.  Switches on when triggered.

Style:
0  - normal
1  - flicker (first variety)
2  - slow strong pulse
3  - candle (first variety)
4  - fast strobe
5  - gentle pulse
6  - flicker (second variety)
7  - candle (second variety)
8  - candle (third variety)
9  - slow strobe
10 - flourescent flicker
11 - slow pulse, not fading to black

   styles 32-62 are assigned by the light program for switchable lights

63 - testing
```

Regular Light:

```
{
    "classname" "light"
    "origin" "X Y Z"    // Tells where the light is
    "light" "#"         // Tells how bright the light is (optional - default 200)
    "style" "#"         // How the light appears
    "spawnflags" "#"    // State light starts in
    "targetname" "#"    // Target id of the light
}
```

Fluorescent Light:

```
{
    "classname" "light_fluoro"
    "origin" "X Y Z"    // Tells where the light is
    "light" "#"         // Tells how bright the light is (optional - default 200)
    "style" "#"         // How the light appears
    "spawnflags" "#"    // State light starts in
    "targetname" "#"    // Target id of the light
}
```

Fluorescent Light (makes sparking sound):

```
{
    "classname" "light_fluorospark"
    "origin" "X Y Z"    // Tells where the light is
    "light" "#"         // Tells how bright the light is (optional - default 200)
    "style" "#"         // How the light appears
    "spawnflags" "#"    // State light starts in
    "targetname" "#"    // Target id of the light
}
```

Torches:

```
{
    "classname" "light_torch_small_walltorch"
    "origin" "X Y Z"    // Tells where the light is
    "light" "#"         // Tells how bright the light is (optional - default 200)
    "style" "#"         // How the light appears
    "spawnflags" "#"    // State light starts in
    "targetname" "#"    // Target id of the light
}
```

Fire:

```
{
"classname" "light_flame_large_yellow, light_flame_small_yellow, light_flame_small_white"
"light" "#"         // Tells how bright the light is (optional)
"origin" "X Y Z"
}
```

2.3.2 Player Movement Entities

Level Change Trigger (attaches to brush):

```
{
    "classname" "trigger_changelevel"
    "map" "mapname"  // Map to change to on trigger (e.g. e1m8)
    "spawnflags" "#" // Flags describing the object (optional)

    {
        <INSERT BRUSH INFO HERE>
    }

}
```

Teleport Trigger (attaches to brush):

```
{
    "classname" "trigger_teleport"
    "target" "t#"     // Teleport destination name to teleport to.
    "targetname" "t#" // Trigger name (optional) - only teleports once
                                             activated if targetname is present.

    {
        <INSERT BRUSH INFO HERE>
    {

}
```

Teleport Destination:

```
{
    "classname" "info_teleport_destination"
    "origin" "X Y Z"
    "angle" "#"       // angle the player will face upon leaving teleport
    "targetname" "t#" // Teleport's trigger name
}
```

2.3.3 Movers

Door (attaches to brush):

```
{
    "classname" "func_door, func_door_secret"
    "angle" "#"      // angle it faces
    "speed" "#"      // speed of movement
    "targetname" "#" // Door's trigger name
    "sounds" "#"     // sound it makes
    "wait" "#"       // delay before closing
    "spawnflags" "#"       // Flags describing the object (optional)
    "lip" "#"        // some kind of offset (optional)

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

Spawns on certain skill levels (controlled by spawnflags), can be removed by a trigger. (attaches to brush):

```
{
    "classname" "func_wall"
    "spawnflags" "#" // flags for something (optional I'd guess)
}
```

A platform (i.e. lift or elevator, attaches to brush):

```
{
    "classname" "func_plat"
    "height" "#" // height it rises? (optional)
    "sounds" "#" // sound it makes (optional)

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

Moving platform (Attaches to brush):

```
{
    "classname" "func_train"
    "sounds" "#"      // Sound it makes when activated
    "speed" "#"       // Speed at which it moves (optional)
    "target" "t#"     // Trigger name of its first path_corner destination
    "targetname" "t#" // Its trigger name
    "dmg" "#"         // Damage done on crush

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

Describes path of train/monsters:

```
{
    "classname" "path_corner"
    "origin" "X Y Z"
    "target" "t#"     // Trigger name of next train destination.
    "targetname" "t#" // It's trigger name.
}
```

2.3.4 Triggers/Switches

A button/switch (attaches to brush):

```
{
    "classname" "func_button"
    "angle" "#"   // Angle button moves?
    "speed" "#"   // Speed it moves in?
    "target" "t#" // Trigger name of target entity
    "health" "#"  // If there is health, button is shootable
    "sounds" "#"  // Sound it makes when activated
                                    // 1 == Steam Metal
                                    // 2 == Wooden Clunk
                                    // 3 == Metallic Click
                                    // 4 == In-Out
    "wait" "#"    // Wait until retrigger? (-1 stays pressed)
    "delay" "#"   // Delay before action is triggered

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

Walk-over trigger (attaches to brush):

```
{
    "classname" "trigger_once, trigger_multiple, trigger_onlyregistered, trigger_secret"
    "style" "#"       // 32 works
    "killtarget" "t#" // Kills something [for triggering monster events] (optional)
    "target" "t#"     // Trigger name of target
    "sounds" "#"      // Sound made when triggered
                                        // 1 == Secret Sound
                                        // 2 == Beep Beep
                                        // 3 == Large Switch
                                        // 4 == Set "message" to text string
    "wait" "#"        // Delay before retrigger. some classes only. (optional)
    "delay" "#"       // Delay before action is triggered

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

Triggers *target* after it is triggered *count* times:

```
{
    "classname" "trigger_counter"
    "targetname" "t#" // Its trigger name
    "target" "t#"     // Trigger name of its target
    "count" "#"       // Decrements on each trigger.  When 0 activates target.
    "wait" "#"        // Required delay before retrigger
    "delay" "#"       // Delay before action is triggered

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

Used to stagger events on a trigger:

```
{
    "classname" "trigger_relay"
    "origin" "X Y Z"  // Where it is located
    "killtarget" "#"  // Removes targeted entity (optional)
    "targetname" "t#" // Its trigger name
    "target" "t#"     // Trigger name of its target
    "delay" "#"       // Delay before action is triggered
}
```

Makes a monster jump when it reaches a brush:

```
{
        "classname" "trigger_monsterjump"
    "speed" "#"       // Forward velocity of the monster jump
    "height" "#"      // How high the monster jumps
    "angle" "#"       // Angle towards which the monster jumps

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

An invisible brush?:

```
{
        "classname" "func_illusionary"

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

2.3.5 Traps/Things harmful to you:

Triggerable Nail-firing trap:

```
{
    "classname" "trap_spikeshooter"
    "origin" "X Y Z"
    "angle" "#"       // Angle the trap fires at
    "targetname" "t#" // Trap's trigger name
    "spawnflags" "#"  // ??? 1024 works
}
```

Constant Nail-firing trap:

```
{
    "classname" "trap_shooter"
    "origin" "X Y Z"
    "angle" "#"       // Angle the trap fires at
    "spawnflags" "#"  // ??? 1024 works
    "wait" "#"        // Time between shots
}
```

Fireballs:

```
{
    "classname" "misc_fireball"
    "origin" "X Y Z"
    "speed" "#" // Tells how fast the fireball moves
}
```

2.3.6 Miscellaneous:

Pushes the player in a direction:

```
{
        "classname" "trigger_push"
        "origin" "X Y Z"  // Where it is located
    "speed" "#"       // Force of the push
        "angle" "#"       // Direction player is pushed towards (-1=up -2=down, other=normal)
}
```

Cameras for the intermission screen:

```
{
    "classname" "info_intermission"
    "origin" "X Y Z" // location of camera
    "mangle" "X Y Z" // location the camera looks at
    "angle" "#"      // angle of the camera
}
```

Setting difficulty level (attaches to brush):

```
{
    "classname" "trigger_setskill"
    "message" "#"   // Skill level to change to.

    {
        <INSERT BRUSH INFO HERE>
    }
}
```

Lightning used to kill the boss of shareware. I'll figure out how to use it later... I pretty much know how, I just want to test it before I put it here :). From the testing I've done, the lightning produced will not damage a player, however. It probably triggers a script to give the boss character damage.

```
{
    "classname" "event_lightning"
    "origin" "X Y Z"  // location of lightning (origin?)
    "targetname" "t#" // It's trigger name
}
```

Section 4: Level Structures

2.4.1 Moving Platforms:

Creating a moving platform isn't that difficult. First, you must define the brush that will do the moving. Here's an example:

```
{
"classname" "func_train"
"sounds" "1"
"speed" "128"
"target" "t1dest1"
"targetname" "t1"
    {
        ( -768 0 0 ) ( -768 1 0 ) ( -768 0 1 ) GROUND1_6 0 0 0 1.0 1.0
        ( -640 0 0 ) ( -640 0 1 ) ( -640 1 0 ) GROUND1_6 0 0 0 1.0 1.0
        ( 0 -384 0 ) ( 0 -384 1 ) ( 1 -384 0 ) GROUND1_6 0 0 0 1.0 1.0
        ( 0 -256 0 ) ( 1 -256 0 ) ( 0 -256 1 ) GROUND1_6 0 0 0 1.0 1.0
        ( 0 0 -256 ) ( 1 0 -256 ) ( 0 1 -256 ) GROUND1_6 0 0 0 1.0 1.0
        ( 0 0 -128 ) ( 0 1 -128 ) ( 1 0 -128 ) GROUND1_6 0 0 0 1.0 1.0
    }
}
```

Now you define each of the path_corners it will travel to. When it reaches a **path_corner**, it will float to the next **path_corner** defined in the **target** tag of the **path_corner**. The platform will start at the **path_corner** pointed to by the platform's **target** tag. It will continue the loop indefinitely and it will go through walls to get to its destination. Here's its **path_corner**s would look like:

```
{
"classname" "path_corner"
"origin" "0 0 0"
"targetname" "t1dest1"
"target"" "t1dest2"
}

{
"classname" "path_corner"
"origin" "0 128 0"
"targetname" "t1dest2"
"target"" "t1dest1"
}
```

2.4.2 Monsters and Triggers:

Have you been wondering how you can get events to trigger when a monster dies, as first seen in E1M2 with the demons? Well, it's not too difficult. When you attach a **target** tag to a monster, the monster's death will trigger the event. I believe (not tested) that if other monsters have a **targetname** tag the same as a monster with the **target** tag, the **target** event will only occur when all monsters with a matching **targetname** tag are dead. The monster with the **target** tag need not have the **targetname** tag.

2.4.3 How to use trigger_count:

The **trigger_count** class is quite an interesting trigger. You know of the area in E1M1 where you have to hit the three switches to open the door? Well, that's done using a **trigger_counter**. Each of the buttons you hit has its **target** property set so it points to a **trigger_counter**. The **trigger_counter** has its **count** tag set to three. Each time a switch is hit, the **trigger_counter**'s **count** property will decrement by one. When it reaches zero, it will open the door. Each button can only be triggered once as it has a **wait** of -1. Here's an example given to me by Remco Stoffer:

```
{
 "classname"    "func_door"
 "targetname"    "door2"
 "target"        "light1"
 "angle"        "-1"
 "wait" "-1"
 "sounds" "4"
 "message"   "press all buttons"
 {
  ( -10  120    0 ) ( -10   80    0 ) (  10  120    0 ) *slime0 2 0 0 1.000000 1.000000
  (  10   80    0 ) (   0   80  100 ) ( -10  120    0 ) *slime0 2 0 0 1.000000 1.000000
  (   0   80  100 ) ( -10   80    0 ) (   0  120  100 ) *slime0 2 0 0 1.000000 1.000000
  ( -10   80    0 ) (  10   80  100 ) (  10   80    0 ) *slime0 2 0 0 1.000000 1.000000
 }
}
{
 "classname"      "trigger_counter"
 "count"          "3"
 "targetname"     "door1"
 "target"         "door2"
 "wait"           "-1"
}
{
 "classname"      "func_button"
 "angle"          "0"
 "wait"           "-1"
 "target"         "door1"
 {
  (  180 -200   50 ) (  180 -180   50 ) (  200 -180   50 ) tlight11 16 0 0 1.000000 1.000000
  (  180 -200   30 ) (  200 -200   30 ) (  200 -180   30 ) tlight11 16 0 0 1.000000 1.000000
  (  200 -180   50 ) (  180 -180   50 ) (  200 -180   30 ) tlight11 16 0 0 1.000000 1.000000
  (  180 -180   50 ) (  180 -200   50 ) (  180 -180   30 ) tlight11 16 0 0 1.000000 1.000000
  (  180 -200   50 ) (  200 -200   50 ) (  180 -200   30 ) tlight11 16 0 0 1.000000 1.000000
  (  200 -200   50 ) (  200 -180   50 ) (  200 -200   30 ) tlight11 16 0 0 1.000000 1.000000
 }
}
{
 "classname"      "func_button"
 "angle"          "0"
 "wait"           "-1"
 "target"         "door1"
 {
  (  180 -10    50 ) (  180  10    50 ) (  200  10    50 ) +0basebtn 0 0 0 1.000000 1.000000
  (  180 -10    30 ) (  200 -10    30 ) (  200  10    30 ) +0basebtn 0 0 0 1.000000 1.000000
  (  200  10    50 ) (  180  10    50 ) (  200  10    30 ) +0basebtn 0 0 0 1.000000 1.000000
  (  180  10    50 ) (  180 -10    50 ) (  180  10    30 ) +0basebtn 0 0 0 1.000000 1.000000
  (  180 -10    50 ) (  200 -10    50 ) (  180 -10    30 ) +0basebtn 0 0 0 1.000000 1.000000
  (  200 -10    50 ) (  200  10    50 ) (  200 -10    30 ) +0basebtn 0 0 0 1.000000 1.000000
 }
}
{
 "classname"      "func_button"
 "angle"          "0"
 "wait"           "-1"
 "target"         "door1"
 {
  (  180  180   50 ) (  180  200   50 ) (  200  200   50 ) +0basebtn 0 0 0 1.000000 1.000000
  (  180  180   30 ) (  200  180   30 ) (  200  200   30 ) +0basebtn 0 0 0 1.000000 1.000000
  (  200  200   50 ) (  180  200   50 ) (  200  200   30 ) +0basebtn 0 0 0 1.000000 1.000000
  (  180  200   50 ) (  180  180   50 ) (  180  200   30 ) +0basebtn 0 0 0 1.000000 1.000000
  (  180  180   50 ) (  200  180   50 ) (  180  180   30 ) +0basebtn 0 0 0 1.000000 1.000000
  (  200  180   50 ) (  200  200   50 ) (  200  180   30 ) +0basebtn 0 0 0 1.000000 1.000000
 }
}
```

2.4.4 Teleporting Monsters:

Unlike in Doom-Engine games, you can precisely teleport monsters into new locations in Quake. To do so, you must first create a out of reach area for the monsters to reside in. Give this area a **trigger_teleport** tag and assign a **targetname** tag to it. Create a **teleport_destination** where you want the monster to appear. Now, you must create a trigger whose **target** property points to the **trigger_teleport**'s targetname. When this trigger is activated, the monster in the room will teleport to the **teleport_destination**. Make sure that there is only one monster per room and one room per **teleport_destination**. Otherwise, when the teleport is triggered, all the monsters will telefrag each other (like what happens in E1M7 when you win).

2.4.5 Properties of Buttons:

The behavior of buttons can be altered in many ways. By default, buttons are activated by pressing them (moving near them). Buttons can be made shootable by giving them a **health** tag. Unless you want to have to shoot the button tons of times, set the **health** tag to "1". If you want to have the button flash when you shoot it, you must include all of the button animation textures in the level. You can just put them on brushes outside the level.
