

ammo_*
ammo_fuel
ammo_homing_missiles
crane_beam
crane_control
crane_hoist
crane_hook
crane_reset
func_bobbingwater
func_button
func_conveyor
func_door
func_door_rot_dh
func_door_rotating
func_door_swinging
func_explosive
func_force_wall
func_killbox
func_monitor
func_pendulum
func_pushable
func_reflect
func_rotating
func_rotating_dh
func_timer
func_trackchange
func_tracktrain
func_train
func_trainbutton
func_vehicle
func_wall
func_water
hint_path
info_notnull
info_player_coop
info_player_deathmatch
info_player_start
item_*
item_flashlight
item_jetpack
key_*
light
misc_actor
misc_deadsoldier
misc_easter*
misc_explobox
misc_insane
misc_strogg_ship
misc_teleporter
misc_teleporter_dest
misc_viper
misc_viper_bomb
model_spawn
model_train
model_turret
monster_*
monster_commander_body
path_corner
point_combat
path_track
target_actor
target_anger
target_animation
target_attractor
target_blaster
target_bmodel_spawner
target_cd
target_change
target_changelevel
target_crosslevel_trigger
target_effect
target_explosion
target_fade
target_failure
target_fog
target_fountain
target_help
target_laser
target_lightramp
target_lightswitch

target_locator
target_lock
target_lock_clue
target_lock_code
target_lock_digit
target_monitor
target_monsterbattle
target_movewith
target_playback
target_precipitation
target_rocks
target_rotation
target_set_effect
target_skill
target_sky
target_spawner
target_speaker
target_splash
target_temp_entity
target_text
tremor_trigger_multiple
trigger_bbox
trigger_disguise
trigger_fog
trigger_hurt
trigger_inside
trigger_key
trigger_look
trigger_mass
trigger_monsterjump
trigger_multiple
trigger_once
trigger_push
trigger_relay
trigger_scales
trigger_teleporter
trigger_transition
turret_base
turret_breach
turret_driver
weapon_*

weapon_blaster
worldspawn

ActorPak
AI and behavior
Console commands
Count
Developer tools
Dmgteam
Fog effects
Footsteps
Gib fade
Homing rockets
MoveWith
Physics
Sniper zoom
Sounds
Third-person perspective
Turn_rider

Made With Lazarus



Maps

Release Date	Title	Author
08/20/2001	Malfunction Junction	Brendon Chung (<i>blender81</i>)
08/11/2001	Damage, Inc.	Jeffrey Hayat
03/29/2001	Divide By Zero	Scott Kraus (<i>SoftShoulder</i>)

Want to see your map here?

If you make a map using **Lazarus** entities and features, the **Lazarus Q2 Mod** team would like to provide a place where your map can be featured and accessed by anyone wishing to play it. We will announce any new maps submitted and accepted here on this website, on the [Rust Q2 forum](#), and will ask Rust newsmonkeys to post a blurb about them on the main [Rust](#) news page. We will mirror such maps on Telefragged's ftp server so that the map author need not secure server space himself for an extended period of time.

So what's the catch? Well, we **do** have a criteria for all submissions that we'd like met:

1. All submissions should be addressed to [us](#) - note that this is for asking us to check out the work, *not so you can mail us the map(s) directly!* We are not on fat pipes so please don't send us any big attachments. Tell us where we can download the map(s) and we'll do so.

We will not consider any maps that aren't submitted, so if you don't want your maps to appear here, no problem - just don't submit them :-)

2. Maps should be of good quality and playable. This means no partially-compiled maps, no buggy maps, etc. No [Cranky Steve](#) material, in other words. We want both your maps and **Lazarus** to look good here. "Playable" not only means that the map will run, but that it plays like a game. This last is obviously subjective.

In the event any of you have heard the words "size doesn't matter", well this time you can believe it. A good, quality map that takes all of 3 minutes to play through is fine, just as a good, quality full-blown 70+meg mission pack is fine too. Bottom line is, we want fun maps that make use of **Lazarus**.

3. No submission should violate copyright laws or anyone's EULA. That means no textures from other games, no redistributed models, skins, sounds, sprites, etc. etc. We know there's a really great propensity to "cheat" just a little bit, and it's really easy to rationalize why such inclusions aren't really hurting anyone. However we'd hate to see anything at all that appears on the **Lazarus** site to be foxed for any reason, so we're standing firm on this. If we find out later that a supposedly "clean" submission that we posted on the website includes something other than royalty-free content, we'll pull it fast, and what's more, we'll be annoyed too.
4. Submissions should be compressed in .ZIP format, and the maps and associated files should be in a pakfile - No Loose

Files Please! For full information on how to set this up, please see the [Redistribution](#) page.

5. We will endeavor to evaluate submissions in a timely manner, but things like **Lazarus** updates and Real Life™ are likely to get in the way, so please don't hassle us if we take a bit of time with this. Also, at this time we have no plans to offer any formal website reviews on any submission - we'll probably just list the map(s) along with the author's name, and a short description of what the map(s) contain. This may change of course.
6. If an author wishes to withdraw his submission, or wishes his posted submission to be removed from the Telefragged server and its description removed from the **Lazarus** website, he need only contact us and it will be done as soon as possible. **Your maps remain your property.**
7. Any links from review sites, home pages, etc. should not link directly to the ftp download link for the map(s) themselves, but instead should link to the **Lazarus** page which features the information about the map(s). This is just to be fair to our webhosts, Rust and Telefragged. This URL will be provided to you when a submission is accepted.
8. Decisions of what submissions are accepted and what aren't, are made by the **Lazarus Q2 Mod** team.

Well, that's the deal. If you're interested, and you've got something to submit, [drop us a line](#).

[Lazarus Main Page](#)



ammo_*
crane_beam
crane_control
crane_hoist
crane_hook
crane_reset
func_bobbingwater
func_button
func_door
func_door_rot_dh
func_door_rotating
func_explosive
func_force_wall
func_killbox
func_monitor
func_pushable
func_rotating
func_rotating_dh
func_timer
func_train
func_trainbutton
func_vehicle
func_wall
hint_path
info_notnull
info_player_start
item_*
item_flashlight
key_*
light
misc_actor
misc_deadsoldier
misc_insane
misc_teleporter
misc_teleporter_dest
misc_viper_bomb
model_spawn
model_train
monster_*
monster_commander_body
path_corner
point_combat
target_actor
target_anger
target_animation
target_attractor
target_blaster
target_cd
target_changelevel
target_crosslevel_trigger
target_effect
target_explosion
target_fog
target_help
target_laser
target_lightramp
target_lightswitch
target_locator
target_lock
target_lock_clue
target_lock_code
target_lock_digit
target_monitor
target_monsterbattle
target_rocks
target_rotation
target_spawner
target_speaker
target_splash
target_temp_entity
target_text
tremor_trigger_multiple
trigger_bbox
trigger_fog
trigger_hurt
trigger_inside
trigger_key
trigger_look
trigger_mass
trigger_multiple

trigger_once
trigger_relay
trigger_push
trigger_scales
trigger_teleporter
trigger_transition
turret_base
turret_breach
turret_driver
weapon_*

ActorPak
Console commands
Count
Developer tools
Fog effects
Gib fade
MoveWith
New sounds
Sniper zoom
Third-person perspective

Features

Entities

[What's New](#)
[Downloads](#)
[Change log](#)
[About Lazarus](#)
[Lazarus Maps](#)
[Install and Run](#)
[Redistribution](#)
[Contact](#)
[Credits](#)

- Lazarus Updates -

[What's New](#)
[Downloads](#)
[Change log](#)
[About Lazarus](#)
[Lazarus Maps](#)
[Install and Run](#)
[Redistribution](#)
[Contact](#)
[Credits](#)

- The Lazarus Manual -

Features

Entities

[ActorPak](#)
[AI and behavior](#)
[Console commands](#)
[Count](#)
[Developer tools](#)
[Dmgteam](#)
[Fog effects](#)
[Footsteps](#)
[Gib fade](#)
[Homing rockets](#)
[MoveWith](#)
[Physics](#)

- Lazarus Updates -

[What's New](#)
[Downloads](#)
[Change log](#)
[About Lazarus](#)
[Lazarus Maps](#)
[Install and Run](#)
[Redistribution](#)
[Contact](#)
[Credits](#)

- The Lazarus Manual -

Features

[ActorPak](#)
[AI and behavior](#)
[Console commands](#)
[Count](#)
[Developer tools](#)
[Dmgteam](#)
[Fog effects](#)
[Footsteps](#)
[Gib fade](#)
[Homing rockets](#)
[MoveWith](#)
[Physics](#)
[Sniper zoom](#)
[Sounds](#)
[Third-person perspective](#)
[Turn_rider](#)

Entities



- Lazarus Updates -

[What's New](#)
[Downloads](#)
[Change log](#)
[About Lazarus](#)
[Lazarus Maps](#)
[Install and Run](#)
[Redistribution](#)
[Contact](#)
[Credits](#)

- The Lazarus Manual -

Features

[ActorPak](#)
[AI and behavior](#)
[Console commands](#)
[Count](#)
[Developer tools](#)
[Dmgteam](#)
[Fog effects](#)
[Footsteps](#)
[Gib fade](#)
[Homing rockets](#)
[MoveWith](#)
[Physics](#)
[Sniper zoom](#)
[Sounds](#)
[Third-person perspective](#)
[Turn_rider](#)

Entities



Lazarus Description



Lazarus is an assortment of entity modifications made to Quake2 over the last year or so. Much of this work was done as part of the (currently) defunct [Tremor Mod](#). It includes a seemingly ever-increasing number of useful changes to standard entities, as well as a nifty array of new whizbang entities, including... well... go click on some links over there on the left.

The basic philosophy behind Lazarus is to give Quake2 level designers an entity set which will allow them to make better, more challenging, more interactive, and more entertaining single-player maps. However, enhancing the gameplay of single-player is not its only goal. As time goes on features that can be used in multiplayer will be added as well, many of which will be available while running any plain-vanilla Quake2 map - not just maps designed with Lazarus in mind. Since the details of this mod are geared towards the level designer, most of the feature descriptions you will find here discuss the properties of the modifications. What these changes mean to the gameplay is left to your imagination - since the best thing that can happen is that users will come up with totally unforeseen ways to make use of all the new stuff.

What Lazarus **isn't** is another mod which presents the mapper with a game with its own unique rules. The intention is not to make Quake2 a different game, but instead to make it a better one. Virtually every Quake2 map could be adapted to make use of the greater options Lazarus offers, without sacrificing a bit of the design of the original version.

This is a work-in-progress, and as such likely contains a few bugs and quirks. Comments, suggestions, and really obnoxious criticisms are always welcome [here](#).

DISCLAIMER

Neither David Hyde nor Tony Ferrara make any warranties regarding this product. Users assume responsibility for the results achieved for computer program use and should verify applicability and accuracy.

[Lazarus Main Page](#)

ActorPak

Lazarus 1.4 introduces a fully-functional **misc_actor** entity that can make use of **any** available player model. Misc_actor allows you to add good guys to help out against the monsters, as well as create entirely new monsters. The largest single drawback to using player models for misc_actor (or any other entities) is that most existing player models will not work as is. For entities other than the player, Quake2 requires that all skins that might be used by a model be referenced within the model itself. In the original game design, this was deemed too restrictive for player models, since that requirement would eliminate the possibility of using custom skins for players. While this was no doubt a good design decision for player models, it leaves something to be desired if a player model is used for any other entity. The solution is to modify player models to include references to all skins that might be used by that model. A model modified in this way may then be used **both** as a normal player model and as the model for misc_actor.

And that's exactly what this page is all about: we've taken many original player models and modified them to include skin references so that you don't have to. You can get the whole thing, wrapped up in a single pakfile, **right here**.

Note: This is important, so pay attention :-). We're fairly happy to be able to add this functionality to Quake2, and foresee all sorts of uses for it. However, it comes at a price. Misc_actor loads up to 13 unique sounds for each unique player model. You cannot have more than 256 unique sounds in a map, or the game will crash with the dreaded **ERROR: Index overflow**. You may have many actors that use the same model, and all of those actors will only account for 13 additional sounds. However, it doesn't take much effort to figure out that if your map already makes use of many unique sounds, the map can't accept a large number of unique player models. So, you say... fine, I just won't distribute the custom sounds with these models and they will all use the standard male sounds. Well that's fine if the end user doesn't already have the player model in question. But **if he does**, his custom sounds will be used even though they are likely in a different folder. The short version of this is that you'll need to playtest your map to ensure you don't get an Index Overflow due to misc_actor sounds.

You'll note the glaring absence here of the standard Q2 male, female, cyborg, and crakhor models. The reasons for this are many, but the bottom line is that you don't need to distribute or modify the male, female, and/or cyborg models in any way. To get around possible copyright problems Lazarus performs a bit of magic under the hood and will modify the end user's male, female, and/or cyborg models to include skin references, and copy those modified versions of the model files to [gamedir]/players. (Thanks to **Argh!** for providing the code that handles this.) This change has the blessing of id Software, so there's no need for you to worry about copyright infringement issues with those models. The short version of this is that if you want all of your misc_actors to use the male, female, or cyborg models, well then you're wasting your time on this page... go make maps!

We've asked for permission from the original authors to redistribute the models described below, and we've received such permission for most of the models, either by direct correspondence with the author or an implied approval in the model documentation. However, we've never received a response from the authors of several of these models. We don't **think** we're stepping on anyone's toes here, but if a model author sees his work here and objects to us redistributing that model, by all means please **let us know** and we will immediately remove the model from the Lazarus pages. Although we'll **show you how** to modify additional player models so that they may be used with misc_actor, we **strongly** suggest that you do **not** distribute models without the permission of the original author.

Following is a description of each of the models included in actorpak.zip. The description includes suggested bounding box coordinates (so your misc_actor doesn't intersect walls or allow projectiles to pass through his body), muzzle offsets for various weapons, and our suggestions for weapons.

The bounding box coordinates (BBox) are only a **recommendation**, and not meant to represent the absolute minimum and maximum coordinates of a model's motion. There's a balancing act here, and there is no such thing as the "correct" bounding box that will both contain the model and block weapon fire only when it should. BBox is sufficiently large to bound all animations other than taunts, jump, and death sequences. **All** player models exceed their bounding boxes in death animations, and most jump through the top limit.


Recommended muzzle offsets are not by any means exact. A couple of new Lazarus developer features will assist you in determining the correct muzzle offset for any weapon with any player model. A procedure for doing this is described on the **developer tools** page.


For misc_actors that switch weapons with distance (e.g. sounds=0703), the muzzle location is used for both weapons. So

again we're looking at a compromise - how to set a muzzle location that looks good for both weapons. This will probably prove to be pretty elusive. If you have to place the muzzle location where one of those weapons looks better than the other, we've found that weapons that emit particles at the firing origin (shotgun, super shotgun, machinegun, and chaingun) require a bit more care than projectile-firing weapons do.

Of course if a model does not have VWEF support, it will only have one muzzle offset that is applicable to all weapons, since there is only one weapon model to worry about.

Here's what's in the **Lazarus ActorPak**:

<div> <div>  </div> <div> <h1>Alien</h1> <p>by AMC</p> </div> </div>	<div>Usermodel:</div> <div>alien</div>
	<div>Skins:</div> <div>4</div>
	<div>Extras:</div> <div>Vwep: No/Sound Pak: Yes</div>
	<div>BBox:</div> <div>bleft: (-28 -28 -24) tright: (28 28 32)</div>
	<div>Firing origin:</div> <div>muzzle: (42 5 15) muzzle2: NA</div>
	<div>Comments:</div> <div> <p>The model's animation is very jerky with continuous-fire weapons. This problem is particularly noticeable with the chaingun. The weapon model looks like an energy device, so the hyperblaster would be our recommendation.</p> <p>Author Approval: Author's whereabouts unknown.</p> </div>

<div>Hunter</div> <div>by Michael 'Magarnigal' Mellor</div> 	Usermodel:	hunter
	Skins:	5
	Extras:	Vwep: Yes/Sound Pak: Yes
	BBox:	bleft: (-24 -24 -24) tright: (24 24 32)
	Firing origin:	muzzle: <div> <div>Blaster: (32 5 15)</div> <div>SG: (36 5 15)</div> <div>SSG: (36 5 15)</div> <div>MG: (38 4 19)</div> <div>CG: (45 4.5 15)</div> </div> <div> <div>GL: (32 5 15)</div> <div>RL: (40 5 15)</div> <div>HB: (41 4 19)</div> <div>RG: (40 4 19)</div> <div>BFG: (42 5 20)</div> </div> muzzle2: NA
	Comments: All weapon models are available. Longer weapons look a bit strange because of the angle the model holds them in. Author Approval: Author's whereabouts unknown.	

<div><h1>Paranoid Marine</h1><p>by Darren 'HitmanDaz' Pattenden</p></div>	Usermodel:	marine		
	Skins:	4		
	Extras:	Vwep: Yes/Sound Pak: No		
	BBox:	bleft: (-16 -16 -24) tright: (16 16 32)		
	Firing origin:	muzzle: <div><div>Blaster: (18 7 10)</div><div>SG: (22 7 10)</div><div>SSG: (22 7 10)</div><div>GL: (24 7 10)</div><div>RL: (26 7 10)</div><div>HB: (18 7 14)</div></div>		



MG: (18 7 12) RG: (28 7 10)
CG: (26 7 16) BFG: (28 7 10)
muzzle2: NA

Comments: All weapon models are acceptable.
Author Approval: YES

Ratamahatta

by Brian 'EvilBastard' Collins



Usermodel: ratamahatta

Skins: 6

Extras: Vwep: No/Sound Pak: No

BBox: bleft: (-20 -20 -24)
tright: (20 20 32)

Firing origin: muzzle: (24 13 10)
muzzle2: NA

Comments: Machinegun is the obvious choice of weapon. Ratamahatta falls almost completely outside the bounding box in death animations, so if used either gib_health should be very small (like -5) or he should be restricted to open areas.
Author Approval: YES

Rhino

by Andreas Möller



Usermodel: rhino

Skins: 3

Extras: Vwep: No/Sound Pak: Yes

BBox: bleft: (-30 -30 -24)
tright: (30 30 32)

Firing origin: muzzle: (29 7 10)
muzzle2: (27 -15 13)

Comments: Use the chaingun. Anything else would be plain weird. One of this model's taunts is a vertical flip that takes him well above the suggested bbox height.
Author Approval: No response, but web site documentation makes it clear we may use this model.

SAS


by Darren 'HitmanDaz' Pattenden


Usermodel: sas


Skins: 1

Extras: Vwep: No/Sound Pak: No


BBox: bleft: (-18 -18 -24)
tright: (18 18 32)


		
	Firing origin:	muzzle: (17 6.5 17) muzzle2: NA
	Comments: Use the machinegun. <div>Author Approval: YES</div>	


Slith by Ryan Butts & Jeramy Cooke 	Usermodel:	slith
	Skins:	4
	Extras:	Vwep: Yes/Sound Pak: Yes
	BBox:	bleft: (-16 -16 -24) tright: (16 16 32)
	Firing origin:	muzzle: <div> Blaster: (32 7 10)GL: (32 7 10) SG: (32 7 10)RL: (32 7 10) SSG: (32 7 10)HB: (12 6 -1) MG: (25 5 -1)RG: (32 7 10) CG: (25 5 -1)BFG: (20 5 -1) </div> muzzle2: NA
	Comments: Very nice job on the visual weapons with this model. HB and BFG are particularly nice. <div>Author Approval: Pending. Readme implies redistribution is OK.</div>	

Terran Marine by Wrath 	Usermodel:	terran
	Skins:	23
	Extras:	Vwep: No/Sound Pak: Yes
	BBox:	bleft: (-20 -20 -24) tright: (20 20 32)
	Firing origin:	muzzle: (42 7 11.5) muzzle2: NA
	Comments: No way a rocket or grenade is coming out of that gun, but anything else would look fine. The weapon extends well out beyond the front of the bounding box, but since the actor most likely won't be attacking walls this shouldn't be a serious problem. <div>Author Approval: YES</div>	


Walker by Roger [666] Bacon	Usermodel:	walker
	Skins:	3
	Extras:	Vwep? No/Sound Pak? Yes
	BBox:	bleft: (-24 -24 -24)

		
		tright: (24 24 30)
	Firing origin:	muzzle: (9 16 7) muzzle2: (9 -11 7)
	Comments: Use the chaingun or machinegun. The low polycount of this model means you can use a bunch in one area with no problem. Since this is a totally mechanical model, NO_GIB is pretty mandatory. The 2 corpse frames are rather tall. Author Approval: YES	

Waste by Stecki 	Usermodel:	waste
	Skins:	1
	Extras:	Vwep: Yes/Sound Pak: Yes
	BBox:	bleft: (-16 -16 -24) tright: (16 16 32)
	Firing origin:	muzzle: <div> <div>Blaster: (12 9 9)</div> <div>GL: (18 9 7)</div> <div>SG: (22 9 9)</div> <div>RL: (26 9 7)</div> <div>SSG: (20 9 9)</div> <div>HB: (26 7.5 8)</div> <div>MG: (11 11 7)</div> <div>RG: (26 9 7)</div> <div>CG: (26 8 8)</div> <div>BFG: (22 11 7)</div> </div> muzzle2: NA
	Comments: Author Approval: YES	

Xenoid by Dan Bickell 	Usermodel:	xenoid
	Skins:	3
	Extras:	Vwep: No/Sound Pak: Yes
	BBox:	bleft: (-18 -18 -24) tright: (18 18 32)
	Firing origin:	muzzle: (20 12 7) muzzle2: NA
	Comments: Default weapon model pretty much dictates the use of an energy weapon. Author Approval: YES	

Zumlin by Rowan 'Sumaleth' Crawford	Usermodel:	zumlin
	Skins:	6
	Extras:	Vwep: Yes/Sound Pak: Yes
	BBox:	bleft: (-18 -18 -24) tright: (18 18 32)

		
	Firing origin: muzzle: <div> <div> Blaster: (22 3 8) SG: (20 2 9) SSG: (20 2 9) MG: (8 5 4) CG: (22 2 4) </div> <div> GL: (20 2 7) RL: (30 2 9) HB: (20 3 2) RG: (26 2 9) BFG: (16 5 -2) </div> </div> muzzle2: NA	
Comments: Machinegun is a bit too short, and GL has a goofy clipping problem. Author Approval: Pending. Readme implies redistribution is OK.		

Enough already! Gimme the ActorPak!

Modifying other models

Neither of us are modellers, and there may be an easier way to accomplish the following... but this is how we did it. All of these instructions are based on the use of [QME](#), by Rene Post. The price of QME is \$25.00 U.S. as of this writing. Although there is a "Lite" version available at no charge, it is crippled in that it will only save 20 animation frames, which of course is entirely unacceptable for player models.

Before you do anything else, make a backup copy of the player model file you will be operating on. The model file you'll want to edit will be baseq2/players/<name>/tris.md2, where <name> is the name of the model. After making a backup copy of this file, open it in QME. In the left pane you should see a tree view with (probably) only one entry called "new skin". If you do not, select the View menu and click on Skins. Some player models may already have one or more internal skin references, but as often as not those references are incorrect, and we'll delete them later. Also, for models that do **not** have any skin references (which will include most player models), QME inserts a blank skin called... you guessed it... "new skin". We can't delete "new skin" just yet, as QME requires a model to have at least one skin at all times. Now, for our skins: On the File menu, select Import Skin. When the Import Skin dialog comes up, ensure that you are looking in the correct folder, which should be baseq2/players/<name>. You **should** see several .pcx files in that folder. These are the skins for the model. Do **not** select one of the weapon skins (always prefaced by "w_"). Click Open. At this point you will **probably** see an "Import Options" dialog. Under "Replace or Insert", check "Insert new skin". The "Crop or Stretch" options are irrelevant. Click OK, and you're done - for this skin. Repeat these steps, starting with File>Import Skin, for each skin that you want to include support for. When finished importing skins, delete any skins that were originally shown in the skin pane (including "new skin"). You can rearrange the order of the skins by clicking and dragging. Save the model and... you're done. You may now use this player model with misc_actor by setting the actor's usermodel key to <name>.

OR... if you're a tightwad like us, be sure and thank Argh! of [ArghRad](#) fame for this no-cost solution: Download NPherno's Skin Tool [here](#). Once NST is setup (unzip, run, and specify the baseq2 dir), you can modify the player models like so: Backup the player model and skins. (for some odd reason NST insists on re-saving the skin files when you link them to the model... this doesn't alter the file, but it's best to play it safe). Open the player model in NST. Assuming it has no internal skins, it will ask if they should be loaded from the base folder - click Yes. For each skin in the drop-down list, select menu Skins/Link. Select the corresponding .pcx file and click Save. Select menu File/Save to finish.

Auxiliary model files

If you only use the undisturbed ActorPak, you'll never need to worry about skin/model/sound locations for misc_actor. But if you want to use your own models and/or strip out the files from ActorPak that you don't want, you should be aware of the procedure used by Lazarus to find all auxiliary files associated with player models. Lazarus goes through pretty much the same steps taken by the game when attempting to find files. For sounds:

1. Look for disk files in <basedir>/<gamedir>/players/<usermodel>. The sound will be loaded with sound/./players/<usermodel>/<wavename>.
2. Search pak9.pak through pak0.pak in <basedir>/<gamedir> for sound/./players/<usermodel>/<wavename>. Yes, that is the actual path within the pak file, and yes, that's very goofy. This is necessary so that the sounds will work

with misc_actor in either a pak file or on disk.

3. Look for disk files in <basedir>/baseq2/players/<usermodel>.
4. Search pak9.pak through pak0.pak in <basedir>/baseq2; for sound/./players/<usermodel>/<wavename>.
5. Look for disk files in <cddir>/baseq2/players/<usermodel>.
6. Search pak9.pak through pak0.pak in <cddir>/baseq2; for sound/./players/<usermodel>/<wavename>.
7. Use sound/player/male/<wavename>.

For weapon models:

1. Look for visual weapon model in <basedir>/<gamedir>/players/<usermodel>.
2. Search pak9.pak through pak0.pak in <basedir>/<gamedir> for /players/<usermodel>/<weapmodel>.
3. Look for visual weapon model in <basedir>/baseq2/players/<usermodel>.
4. Search pak9.pak through pak0.pak in <basedir>/baseq2 for /players/<usermodel>/<weapmodel>.
5. Look for visual weapon model in <cddir>/baseq2/players/<usermodel>.
6. Search pak9.pak through pak0.pak in <cddir>/baseq2 for /players/<usermodel>/<weapmodel>.
7. Repeat the above, searching for weapon.md2 rather than a visual weapon model.
8. Use players/male/weapon.md2.

Lazarus Main Page

AI and Behavior

One of the biggest gripes with Quake2 was that the game was too easy, and the monsters too dumb. We've been trying to change that, and while we're pretty happy with what we've done so far, we intend to continue to improve things where we can. Most of these changes affect all maps when run using the modified **Lazarus** game .dll - even those of the original game. Others changes will only be available in new maps designed to use those features.

This document will also discuss, to some extent, the behavior of the `misc_actor`. There's more to how actors behave than this though, and that info can be found on the [misc_actor](#) page. Likewise, extra options for monsters in new maps are documented on the [monster](#) page.

Other changes that have been made are [monsterclass](#)-specific. What those changes are will be noted where appropriate.

Self-preservation

We don't like to watch monsters suffer needlessly (we want to **make** them suffer), so **Lazarus** gives monsters (and actors) a temporary entity to get mad at and chase to a safer area when they are damaged by environmental effects. For example, in standard Quake2 if a monster happens to be standing in a rather bad spot when a `func_water` lava brush rises up to fry him, he will continue standing in the same spot, contemplating what to have for dinner. In **Lazarus** this same situation will result in the monster attempting to go elsewhere - hopefully exiting the lava in the process. This means no more monsters standing in place while doors bounce off their heads, or simply groaning while taking no action while being crushed by other moving brush models.

To see the above-mentioned temporary entity (which we have fondly dubbed as "thing"), set *developer* to 1 and watch for a glowing Carmack head to appear when a monster/actor is hurt by the environment.

In a related way, monsters/actors now understand lasers. One of the goofiest things in the original game was how you could lure a monster into charging mindlessly through lasers - after all, these are the guys that supposedly set the laser barriers, so they should know they're there. Savvy mappers could make clip brush `func_walls` which toggle with the lasers for their own maps, and this would serve to keep the monsters out of trouble. But this solution means the player can't use the knockback of a well-placed shotgun blast to push a monster into the lasers either. **Lazarus** makes monsters respect lasers and causes them to endeavor to avoid them. And, if they happen to blunder into low-damage lasers, they'll act to get out of them quickly.

Also closely related is a new respect for grenades. No longer will a gunner fire a grenade at you and immediately chase after you, wading through his own live grenades lying on the floor. In short, monsters will never intentionally walk or run into the damage radius of a live grenade.

Evasive action

Monsters and actors whose initial health is greater than 400 will attempt to run away from live grenades on the ground. In techno-speak, during every 0.1 second frame the game checks for grenades on the ground within the grenade's damage radius of a monster or actor. If one is found, the code looks for a good escape route for that monster/actor. Since monsters normally only move at yaw angles that are multiples of 45 degrees, the code only looks along those yaw angles so that the monster won't end up wasting time turning while running away. For those 8 angles, the code finds the direction that will get the monster outside the grenade's damage radius in the shortest distance. In plain English, if there's a good escape route running away from the grenade, the monster will take that route before he'd take a route that makes him run towards (and past) the grenade. Once he reaches that spot he'll turn towards the grenade owner and get mad at him. He won't move from that spot until the grenade explodes. That's true even if another grenade is hurled his way. (We've found that monsters are so slow, both in reaction time and running speed, that taking 2nd and 3rd grenades into account is almost always fruitless). Why the health discriminator? Again, it's a speed issue. Tougher monsters tend to be slower, to the point where you could hold off a tank indefinitely by lobbing grenades at him. In this case the game would actually be **easier** if tough monsters evaded grenades, as they would never have time to fire at the player.

Similarly, monsters and actors will also avoid rockets in flight, **if** the rocket is more than 1000 units away when

detected. Attempting to avoid closer rockets is a waste of time because monsters are simply too slow to get out of the way. When looking for a good escape route, the code ignores angles that are +/- 15 degrees towards or away from the rocket's start point. This is a shortcut to prevent a monster from running right into the rocket's business end or getting one in his back.

The following applies to **misc_actors only**.

An actor will not simply stand around after firing his weapon. He'll run to find some cover if there's any in his vicinity. (He won't take cover that's behind him, since he'd be wasting time turning around if he did). He'll then pause in his new location for an interval of 0-6 seconds, after which he'll resume his attack. Also, actors with low health (relative to their adversary) will try and look for cover, but they won't resume their search for their enemy once they get there. Of course if you go looking for them, they'll be more than happy to shoot back once you find them.

If for any reason you would like to disable actor evasive actions, it can be done with the **actorscram** (for normal evasive maneuvers) and the **actorchicken** (for run and hide maneuvers) console variables. These can be set in the copy of default.cfg you would include with your map distribution.

Jumping

Some monsters, and all actors, are not confounded by low obstacles - they'll now simply jump right over them. When jumping, the monsters/actors will use their running animations, which doesn't look too shabby. To prevent them from looking silly, we've limited things so they will not jump on surfaces which have a steeper angle than 30 degrees from the horizontal, and they will not jump on players, or other monsters/actors.

This ability constitutes a pretty big change to the standard game. As such we'd appreciate any **comments** you may have, since we hate making changes that break maps. If for any reason you wish to disable monster or actor jumping for your map, it can be easily done with the **monsterjump/actorjump** console variables, respectively. These can be set in the copy of default.cfg you would include with your map distribution.

Only the following entities jump. Their individual jump-up and jump-down distances are noted below.

	jump-up distance	jump-down distance
misc_actor	48	160
monster_berserk	48	160
monster_gunner	48	48
monster_infantry	48	160
monster_soldier	48	160
monster_soldier_light	48	160
monster_soldier_ss	48	160
monster_parasite	32	160

Pathfinding

Other changes involve fixing a couple of bugs when monsters use pathing nodes. In standard Q2, if a monster is patrolling between path_corners, and one of those path_corners has a "wait" value greater than 0, when the wait expires the monster will start walking in the same direction he's currently facing, and not towards the next path_corner in the sequence. The problem is now fixed. Also in standard Q2, if a monster's combattargeted point_combat is placed so that its direction is at 180 degrees from the monster's position, when the monster becomes angry, the monster will take off in the direction of 0 degrees instead, run until it reaches an obstruction, and only then will it realize its mistake and turn around. This too has been fixed.

As an aside, if you like using pathing nodes to make your monsters look smarter, be sure to check out the **hint_path** entity.

Visibility Recognition Distance

The standard game limits monsters (and therefore misc_actors as well) to a maximum visibility recognition distance

of 1000 map units. Beyond that distance, monsters can't see you, nor will they shoot at you. In maps which include large areas, this limitation can become painfully apparent when you see a bad guy in the distance who doesn't see you. You can snipe at him with the blaster, which pisses him off but beyond running around, he'll do nothing. While this is nice for conserving ammo, it's not fun. **Lazarus** increases this distance to 1280 units (however hearable distance is unchanged). Further customization is possible by setting this property monster-by-monster in new maps, using the monster/actor **distance** keyvalue.

Actor Speed and Accuracy

The following applies to **misc_actors** only.

Lazarus attempts to scale actor abilities with skill level and alignment (good or evil). The maximum speed of the player under normal conditions is 400 units/second, so the misc_actor's maximum speed is the same. However his speed is cut back to a maximum of 300 units/second if he's a bad guy (**BE_A_MONSTER**). This gives a player low on health a chance to get away and take cover, and is pretty much in line with other monster movement speeds.

Weapon accuracy is modified as well, using the single-player skill level (skill cvar) as the modifying element. As skill level gets harder, the weapon accuracy of an actor diminishes. This is so the player doesn't find the actors to be quite as much help at hard skill as a player at easy skill would. Conversely, **BE_A_MONSTER** actors have their accuracy increased as skill level increases. The formula used is as follows:

bad guys: $\text{accuracy} = \text{skill} + 2$
good guys: $\text{accuracy} = 5 - \text{skill}$

If you're interested in the math behind what "accuracy" is, here's an explanation. Accuracy=5 is perfect aim. This means on easy skill, a non-monster actor will never miss (assuming his target doesn't move out of the way). In the same way, if skill=3 is manually set by the player, he'll have monster actors with perfect aim to contend with. For all accuracy levels less than 5, the aimpoint error is roughly:

$[(5 - \text{accuracy}) / 20] * [\text{dist}]$
...to a maximum of:
 $[5 - \text{accuracy}] * [12.8]$

So if target is 200 units away and (after accounting for skill) accuracy=2, the maximum targeting error is 30 units. If the target is a player (with a bounding box of 32x32x56 units), the target will be hit approximately 53% of the time (16/30).

Now what happens if an actor has both **GOOD_GUY** and **BE_A_MONSTER** spawnflags set? Let's assume such an actor in a game where skill=2. Since the **GOOD_GUY** flag is dominant, the actor will move at a max speed of 400 units/second, and his accuracy will be 3. If the player shoots this actor, the **GOOD_GUY** flag is removed and the **BE_A_MONSTER** flag takes over. The actor's speed drops to 300, but his accuracy will increase to 4.

Grenade-firing Entities

For the purposes of this description, we'll refer to gunners, but this stuff applies to grenade-launcher-equipped actors and grenade-firing **turrets** as well. Gunners are much better grenadiers than in any previous version of Q2. They will now always check to ensure that a grenade can hit their target, and choose the machinegun if not. If a grenade **can** hit the target, they use the correct elevation angle to get the grenade there... this change makes the gunner a much more formidable opponent. Some details:

- If the gunner's elevation is at or above the elevation of his target, he'll aim at the target's feet to maximize the probability of splash damage.
- If the gunner is below the target, he'll aim at the center of the target to help minimize the risk of the grenade's randomness causing it to hit and bounce off the edge of the platform that the target is standing on.
- This is cool... gunner will sometimes lead a moving target. His tendency to lead is relative to skill level - on easy he'll lead 20% of the time; on nightmare 65% of the time. The randomness in this decision makes it impossible to predict what the gunner will do.
- If the gunner is at the same elevation as or up to 160 units above the target, he checks to make sure that his "perfect" trajectory would not hit a solid object before reaching the target. In other words... if a gunner is on one side of an open doorway with the player on the other side, and the calculated trajectory would cause the grenade to hit above the door, the gunner adjusts his aim down to get the grenade through the door, hoping for a good bounce.
- The current sv_gravity setting is taken into account, both when checking to see whether grenades can reach

the target and when calculating a good trajectory.

These changes make placing grenade-firing entities in areas **below** the level of the player a real possibility - since they won't just blow themselves up as we've often seen gunners do.

Rocket-firing Entities

Several changes, applying to all rocket-firing monsters/actors:

- Monster/actor will no longer fire suicidal rocket blasts into adjacent walls when the player strafes out of view.
- Monster/actor will lead targets when firing rockets. As with the gunner, the tendency to lead the target is tied to skill level: Easy=20%, Normal=35%, Hard=50%, Nightmare=65%. This does not apply if **homing rockets** are used.
- If the target's feet (or minimum z bounding box coordinate) is below the monster's/actor's launcher, a monster will fire at the target's feet 2/3 of the time, while an actor will fire at the target's feet roughly 1/2 of the time. In all other cases the monster/actor will fire at the target origin.
- Rocket speed is skill level dependent ($500 + 100 * \text{skill}$). That's a rather speedy 800 units/sec on nightmare. Again, this change does not apply to homing rockets.
- Size of the splash damage radius of rockets fired by monsters and evil actors is also skill level dependent, so rockets exploding nearby are more lethal when playing the harder skill levels.

Many of the rocket changes are from Rogue's "Ground Zero" mission pack.

Monster_chick

Iron maidens play 13 animation frames when they decide to fire... at 1 frame every 0.1 seconds, that's a glacial 1.3 second windup time. No wonder they're such pushovers! Coupled with the above rocket-firing changes which prevent rocket-firing when an obstruction exists between the monster and its target, getting a chick's attention and then strafing behind cover meant that she seldom if ever actually fired. The following attempts to make her a bit more formidable:

- The windup sequence is a bit faster on skill levels higher than "easy". Out of the 13 frames, 2 frames are skipped in normal, and 5 frames are skipped on hard and nightmare.
- If her enemy is not visible at end of chick's windup phase, rather than abandon firing, she'll now fire at the feet of her enemy's last known position. Since monster-fired rocket splash damage is increased for the harder skill levels, evading a chick's rocket by ducking into a shallow alcove is a less viable choice.

Monster_medic

Medics have never seen their potential realized, given the rather obtuse AI of Quake2 monsters, and their propensity to fight among themselves. We've made quite a few alterations to try and change that:

- The medic now considers looking for monsters to heal to be a higher priority than before - no more fighting to the death with the player while dead allies littering the landscape are ignored. He will consider any monster in his line-of-sight up to 1000 units away, and he'll run to them to get the job done (within 400 units) rather than waste time with mincing baby steps.
- He will ignore monster corpses if visible yet blocked by a solid - no attempts to heal monsters on the other side of windows.
- Once a Q2 monster becomes mad, either by seeing the player, the player shooting him, or from being triggered when the player is the activator, he's mad at you for life (or until he has another enemy), and will never stop searching for him, no matter how well his enemy hides. A medic may act differently if a **hint_path** is visible to him. If he has been triggered, and can't see the player, after a short time he'll give up on looking for the player and start looking for **hint_paths** instead. As he follows a hint_path chain, he may discover monster corpses, and if he does, he'll attempt to heal them. Upon reaching the end of a hint_path chain, he'll retrace his steps - unless there's a different hint_path chain that he can see (within 512 units), in which case he'll follow that. Carefully-laid hint_path chains can enable medics to work behind the scenes to repopulate a map, undoing all of the player's good work. (You can test the medic's hint_path usage in your map with the **medic_test** developer tool).
- A medic stymied in his attempt to reach a monster corpse due to obstructions will search for **hint_path** chains that will lead him to that corpse. To facilitate this, medics will not break off from following hint_paths as soon

as the corpse is visible; they stay the course until they come within healing range. If none are found, the corpse is temporarily flagged as unhealable, and the medic will move on.

- Medic will no longer attack another monster, even if the other monster deliberately attacks the medic. This avoids a bit of Q2 silliness - medic killing a monster then immediately healing him.
- Medic will no longer attempt to heal a dead monster if that operation would result in the monster and medic bounding boxes intersecting... which makes a bit of a mess when they get tangled up and neither can move. Likewise, he won't attempt to heal a corpse the player is standing on, for similar reasons.
- A check is in place to ensure that a clear line-of-sight exists between medic "muzzle" and the dead monster before initiating "cable attack". This is **not** foolproof, since medic animations cause the cable origin to move around a bit, but it should prevent a few occurrences of medics attempting to heal monsters when the cable can't get there.
- This is a visual tweak: Medic's cable now originates at the correct location. You've probably noticed in standard Q2 that the start point of the cable is fouled up if the medic is close to the dead monster he's attempting to heal.

Coupled with the use of the **NO_GIB** spawnflag for monsters for him to resurrect, the medic can now become the most important strategic target to take out. If the medic has his **IGNORE_FIRE** spawnflag set, you won't even be able to hope for another monster to get mad at him and take him out for you.

Monster_soldier, monster_soldier_light, monster_soldier_ss

Weapon accuracy of the 3 grunt types is now skill level dependent. On easy skill, their accuracy is the same as before, while nightmare skill gives them perfect aim. Another change applies to the light guard only: the speed of his blaster bolt is now determined by skill level as well. On easy skill his bolt travels at the normal 600 units/sec. An extra 100 units/sec is added for each increase in skill level - that makes his bolt a sure-shot 900 units/sec on nightmare.

Lazarus Main Page

Ammo_*

Lazarus adds a few notable enhancements to all ammo_* point entities, which can allow for their more realistic placement in the map as well as making them destroyable by weapon fire. And, **movewith** support is also included. The entities affected are:

- ammo_bullets
- ammo_cells
- ammo_fuel
- ammo_grenades
- ammo_homing_missiles
- ammo_rockets
- ammo_shells
- ammo_slugs

Please see the descriptions for the **NO_DROPTOFLOOR** and **SHOOTABLE** spawnflags for more details.

Legend:
Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0.

delay

Specifies the delay in seconds before **target**, **killtarget** and **message** will fire after the ammo is touched by the player. Default=0.

health

Specifies hit points before the ammo will explode if the **SHOOTABLE** spawnflag is set. Default=20.

killtarget

Targetname of the entity to be removed from the map when the ammo is touched by the player.

message

Specifies the character string to print to the screen when the ammo is touched by the player.

movewith

Targetname of the parent entity the ammo is to **movewith**.

target

Targetname of the entity to be triggered when the ammo is touched by the player.

targetname

Name of the specific ammo entity.

team

Team name of the specific ammo entity. This is only relevant when deathmatch=1. When multiple pickups have identical team names, the first to appear in the .map file will be the one that appears on map load. This pickup entity serves as the master, and its teammates are the slaves. When the master

pickup entity is picked up by a player, any of the teammates will spawn on a randomly rotating basis. Ammo entities may be teamed with other ammo, **items**, and/or **weapons**.

Spawnflags

NO_DROPTOFLOOR Spawnflag (=8)

The ammo_* entity will remain wherever it was placed in the map editor. This allows for placing the entity where it could be left hanging in the air, and/or it could be placed so that its bounding box intersects any solid brush, for example snugging the ammo up against a wall or laying it on its side. The normal 32x32x32 pickup "field" would still be in effect, so care must be taken so that the situation where the ammo can be picked up from the opposite side of a wall or similar is avoided. The ammo will not be lit if the entity's origin is buried in a world brush or otherwise hidden from light. The runtime "DropToFloor:" error diagnostics will not apply if this spawnflag is set. Please see [these tips](#) on making use of this feature easier.

SHOOTABLE Spawnflag (=16)

The ammo_* entity will be destroyable by weapon fire, as determined by **health**. The downside to using this option is that the ammo must be made solid. This is no problem if you can pick the ammo up... in the game, there will be a barely noticeable bump as you run into the model. But if you're already stocked up on the ammo, you won't be able to pick it up, **and** the ammo will block your path... not that this is a serious problem, but it's not what we're accustomed to. Naturally, destroying the ammo will clear the path, but the loss of the ammo could have consequences all its own. When deathmatch=1, a destroyed ammo entity will respawn in 30 seconds.

NOT_IN_EASY Spawnflag (=256)

The ammo_* entity will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The ammo_* entity will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The ammo_* entity will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The ammo_* entity will be inhibited and not appear when deathmatch=1.

Ammo_fuel

The ammo_fuel is a new **Lazarus** point entity that provides power for the **jetpack**. Each ammo_fuel provides 500 fuel units. The maximum fuel-carrying capacity of the player is 1000 units, or 1500 with a bandolier, or 2000 with a pack.

To avoid forcing mappers to distribute unnecessary files, ammo_fuel, like item_jetpack, cannot be obtained via the "give all", "give ammo" or "give fuel" cheats unless developer=1.

Please refer to the **Ammo_*** page for a full list of all keys and spawnflags which can be used by ammo_fuel.

As outlined in the **redistribution** doc, ammo_fuel requires these files:

- [models/items/ammo/fuel/medium/tris.md2](#)
- [models/items/ammo/fuel/medium/skin.pcx](#)
- [pics/a_fuel.pcx](#)

*The fuel ammo model distributed with Lazarus is courtesy of **Privateer** (Paul Rogers).*

[Lazarus Main Page](#)

Ammo_homing_missiles

The ammo_homing_missiles is a new **Lazarus** point entity that works with the standard rocket launcher (or, more accurately, gives the **appearance** of working with the standard RL). When fired, homing missiles track whatever damageable entity the player was aiming at. If no damageable entity is found, homing missiles act identically to normal rockets.

If your inventory includes both standard rockets and homing missiles, you toggle between the use of homing missiles and standard rockets each time you "use rocket launcher" (normally 7 is bound to this command). You cannot fire a homing missile while a previously fired missile is tracking a target. In other words you cannot fire a 2nd missile until the 1st missile explodes.

Note: Because of the unusual method used to select homing missiles, it is **essential** that you include a note to this effect in the text file accompanying your map(s).

As mentioned on the [homing rockets](#) page: *We think if you use homing rockets against monsters in your map it had better be a hellacious (technical term) battle, or it will be just plain unfair. Monsters don't know about or understand homing rockets, and they will be your basic sitting ducks.*

As with most Lazarus custom pickup models, ammo_homing_missiles cannot be obtained via cheat unless *developer* is non-zero. This is done so that custom models, pics, and sounds do not need to be distributed unless they are actually used in a map.

For those who don't like being chased by homing missiles, check it out - homing missiles are the ultimate anti-homing missile missile.

Please refer to the [Ammo_*](#) page for a full list of all keys and spawnflags which can be used by ammo_homing_rockets.

As outlined in the [redistribution](#) doc, ammo_homing_rockets requires these files:

- [models/items/ammo/homing/medium/tris.md2](#)
- [models/items/ammo/homing/medium/skin.pcx](#)
- [models/weapons/v_homing/tris.md2](#)
- [models/weapons/v_homing/skin.pcx](#)
- [pics/a_homing.pcx](#)

[Lazarus Main Page](#)

Lazarus Beta



Beta versions of the **Lazarus** game .dll are occasionally released between official releases, but we do not publicize them on this site. However beta-testing to expose bugs and get gameplay feedback is valuable, and leads to better-quality official releases. If you would like to help us out in this area, then:

Sign up for our Beta-test mailing list!

No promises on the frequency of updates, but if and when they are available, you'll be notified by e-mail where to download the new stuff.

If you're on the mailing list already, and you want to get off of it, then:

Remove me from the mailing list!

DISCLAIMER

Neither David Hyde nor Tony Ferrara make any warranties regarding this product. Users assume responsibility for the results achieved for computer program use and should verify applicability and accuracy.

[Lazarus Main Page](#)

Lazarus Change Log



September ??, 2001
Version 2.1

- New Features -

Func_reflect allows you to create the illusion of mirrored surfaces in the floor, ceiling, or walls of a room. Hats off to **psychospaz** for making his floor reflection code (upon which this is based) public.

Working **func_conveyor**. Moves players, **func_pushables**, and all point entities that use a model and are affected by gravity. See the **conveyor** example map.

Lazarus **footstep** code can now use an external file to determine which textures result in what footstep sounds. So new footstep sounds may be used in **any** map, and WorldCraft users won't have to jump through hoops to add new surface flags to their maps. Also added force field footstep sounds and SURF_FORCE surface flag. Download the **footstep** sounds and the **steps** example map.

New **model_turret** is identical to **turret_breach**, but uses an .md2 model rather than a brush model. See security camera and gun turret examples in the **turret2** map.

Misc_actors now use a proper muzzle flash for shotgun, super shotgun, machinegun, and chaingun. Also, new muzzle offsets are hardwired into the code for id player models using machinegun, chaingun, or hyperblaster (standard offsets were off a bit, which you don't notice in the normal game because DM players don't use a proper muzzle flash).

Monsters can now have their **item** set to item_health_small, item_health, item_health_large, or item_health_mega.

MOVETYPE_TOSS and MOVETYPE_BOUNCE entities (pickups items, debris, gibs, grenades) will now usually bounce off of a steep incline rather than coming to rest. Although this change was made to help prevent pileups with conveyors, it should also prevent some of the silliness you sometimes see with gibs and debris.

Changed **SHOOTABLE** pickup items such that they are now clipped against monsters, actors, and insanes (in addition to players). This change was made primarily to help prevent traffic jams and embedded entities on the new working...

- Other Changes -

Pickup items that normally "droptofloor" are now dropped to the correct location instead of being embedded in the floor by 1 unit as they are in standard Q2 and all previous versions of Lazarus. This error became apparent when working on floor reflections with **func_reflect**.

Geek stuff: "noclip" now toggles player's solid state as well as movetype. So if player is MOVETYPE_NOCLIP, he's also SOLID_NOT rather than SOLID_BBOX. This is mostly useful for cheating while testing stuff. In standard Q2 a noclip player will still "touch" many moving entities. In this version of Lazarus... he won't.

- Squashed Bugs -

Fixed problem with killtargeted dead monsters and monster count. Lazarus only includes monsters that are physically present in a map in the monster count. Spawned monsters aren't counted until they actually spawn into the game, and

killtargeted monsters are removed from the count... trouble is Lazarus also removed **dead** killtargeted monsters, which had the curious effect of showing total killed monsters greater than the total monster count. Oops.

Fixed a problem that prevented custom **footstep** sounds from playing when the player was walking on a slope. Thanks to james pants for pointing out the problem.

August 20, 2001
Version 2.0

- New Features -

Monster **AI changes** - monsters will run away from grenades and attempt to move out of the path of flying rockets.

New BOUNCE, FIRE_ONCE, and START_FADE spawnflags for **target_precipitation** and **target_fountain** can be used to modify the behavior of those entities.

Misc_actor can now be used as a remote-controlled robot when targeted by a **func_monitor**. Move forward/back, jump, crouch, and fire weapons, all from the safety of a remote viewing station. You also have the choice of viewing the world from the eyes of the robot or (not quite as realistic but perhaps easier to control) from a third-person perspective.

Custom debris models from *venomus* are now used by destroyed **func_door**, **func_door_rotating**, **func_explosive**, and **func_pushable**. Choose from metal, glass, barrel, crate, rock, crystal, mechanical, wood, and tech models. The crate model provided has 5 skins. All models have skin references for up to 8 skins, so if you want to create your own custom skins for debris models... well go right ahead :-)

Player-fired homing missiles. This actually uses a completely new weapon, but there's a bit of subterfuge going on in the code to give the appearance of using the standard rocket launcher with new ammo. "use Rocket Launcher" (normally bound to the 7 key), toggles between weapon_rocketlauncher and weapon_hml, assuming the player has **ammo_homing_missiles** in his inventory. As with other custom Lazarus models, ammo_homing_missiles cannot be obtained via give cheat unless *developer* is non-zero. So if you don't want players to have homing rockets, well then don't put them in your map :-). For more info see the **ammo_homing_missiles** page.

New **item_freeze** allows the owner to, in effect, stop time for up to 30 seconds. Monsters are frozen, projectiles stop in mid-air, buttons and doors cannot be opened, etc. Thanks to *venomus* for the swell pickup model.

Several changes to **misc_actor**:

- You may now "+use" **GOOD_GUY** actors much the same as you do in Half-Life. Looking at an actor and pressing +use toggles his follow-the-leader state.
- In previous versions, GOOD_GUY actors would come to the defense of the player only if there was a line-of-sight between actor and player. This requirement has been relaxed such that actors in the player's PVS will now come to the player's defense, so actors may run around a corner or 2 to help out.
- The previous +use = "get out of the way" command is now accomplished with the **go** console command. For best results you'll want to bind a key to this command, and of course explain this in your map's text file.
- Actors don't mind wading in water (but will not swim). Actor will not hesitate to enter water up to his eyeballs (his viewheight). Note that this may result in submerged actor models for short player models, but is generally the correct distance for the id models and most others.

Added ALERTSOUNDS flag to worldspawn **effects** flag. If set, monsters are alerted to player footstep sounds.

Added CORPSEFADE flag to worldspawn **effects** flag. If set, monster, actor, and misc_insane corpses begin to fade and sink into the floor after 20 seconds. You can test this flag in **any** map by setting the **corpse_fade** cvar to 1. The delay before the fade may be controlled with the **corpse_fadetime** cvar. If CORPSEFADE is set, monster_medics are removed from the game at startup.

Added JUMPKICK flag to worldspawn **effects** flag. If set, player can hurt monsters and actors and kick out func_explosive windows, for example, by jumping into them.

Added GOODGUY spawnflag to **turret_breach**. If set, turret_breach will track and fire at monsters. To prevent the turret from cleaning out a map area at startup, GOODGUY turrets are disabled at startup and must be triggered to begin tracking monsters.

Several changes to **func_pendulum**: Improved impact physics, new health and noise values.

Added health key to **info_player_start**. If non-zero, specifies the player's health when he spawns into the game at this location. This is a maximum value (player's health will never **increase** because of it). Why would you want to do this? Say you've just survived (barely) a plane crash at the start of a map...

Added **target_skill**, which allows you to build hub maps in which the player has a choice of skill level to play.

Added EXPLOSIONS_ONLY spawnflag to **func_explosive**, which, if set, makes func_explosive invulnerable to projectiles.

- Other Changes -

Removed the feature of adding player velocity to initial velocity of grenades. While technically correct, this never felt right.

Icon for **flashlight** is now displayed properly when the flashlight is picked up.

- Squashed Bugs -

Several versions ago a NOGIB spawnflag was added to **trigger_hurt**. This feature works by limiting the damage a player takes. However, a problem occurred on Easy skill: damage to the player is cut in half in the generic damage routine, so that NOGIB trigger_hurt would cause the game to creep up on killing the player rather than whacking him in one frame. NOGIB trigger_hurt now takes skill level into account before determining damage.

Func_door with non-zero accel/decel values did not act properly as **movewith** parents. Thanks to *Face the Slayer* for the heads up.

GOODGUY **misc_actors** that were defending the player would become mad at TRIGGER_SPAWN monsters who had not spawned into the game yet, resulting in the actor running in place and looking a bit foolish.

Misc_actor aiming was fouled up with supershotgun, unless the target was at the same elevation as the actor.

May 12, 2001
Version 1.9

- New Features -

Surface-specific **footstep** sounds are now possible. Shallow water, wading through deeper water, and ladder climbing sounds will be used if the worldspawn **effects** STEPSOUNDS flag is set. Other footstep sounds are played when running across brush surfaces with new surface flags: metal, grate, tile, dirt, grass, snow, carpet, and ventilation duct sounds are provided. See the steps.bsp example map for usage.

Target_precipitation allows you to add falling rain, snow, leaves, or a user-defined custom model to your map. **NOTE:** We're happy with this entity, but if abused it can very easily create overflow errors. You cannot create an intense rainstorm that covers more than a very small area. Attempt to do so at your own peril :-). The effects used in the *rain* example map are generally safe (used one at a time). You'll need to thoroughly test your maps under adverse conditions before releasing to ensure your map doesn't fall victim to *SZ_GetSpace: Overflow*.

Please note: If you previously downloaded version 1.8.3, we've made a couple of changes to target_precipitation that you'll need to be aware of. "Count" now specifies the number of models generated per second for all types, rather than models per frame. Also the "mass" value is now used only for user-defined styles; "mass2" specifies the number of splash particles. Sorry for any inconvenience this causes, but these changes are for the better.

If you've taken a look at target_precipitation from version 1.8.3, we invite you to look again for a new feature - the user-defined style opens up a lot of possibilities; all you need is the right model. Want a meteor shower that gives headaches to players and monsters? Well now you can have it.

Target_fountain is functionally very nearly identical to a user-defined style for target_precipitation. The principal difference is that the models all emanate from a point source rather than being distributed across an area.

Point_combat has a new DRIVE_TRAIN spawnflag that causes a monster touching the point_combat to drive the **func_tracktrain** at that location. He'll run you down if he can, and reverse course once he loses sight of you. There are also a few more tracktrain changes that facilitate scripted "lose control of the train and fall in a hole" sequences ala *On A Rail*.

Both of these new features are demonstrated in the updated track1 example map.

Another Jed suggestion: `misc_deadsoldier` may now use any of the standard male player model skins (although without blood).

From Face the Slayer: `func_door` and `func_door_rotating` may now be destroyed. Giving doors a negative health value tells the game that the door will absorb the absolute value of "health" damage points before being destroyed. `Func_areaportal` triggering is taken care of automatically, naturally.

From blender81, monsters now sport new "gib_type" and "blood_type" values. If gib_type is non-zero, the code replaces "gibs" in gib model name with "gib1", "gib2", etc. Models are **NOT** supplied with Lazarus.

Added "effects" value to `worldspawn` to control several global changes that just don't seem to fit anywhere else.

`Target_rocks` has been modified to accept new models (you supply the models) by way of its style value.

A couple of changes have been made to `target_playback` to enhance our 3D sound capabilities. First, a new SAMPLE spawnflag has been added that will cause the game to use FMOD's sample functions rather than stream functions. This spawnflag is only meaningful for .MP3 files, since all other file types automatically use the sample functions for 3D sound. So what does this mean? FMOD requires the use of sample rather than stream functions in order to make use of the sound attenuation features (`target_playback` distance value). So if you have a moderate sized (**not** a multi-megabyte soundtrack) .MP3 file, you can now use all of FMOD's 3D features with it. One of the thunder sounds used in the rain example map makes use of this feature. Secondly, all SAMPLE sounds are now precached at startup. So you'll no longer have an annoying burp in the game as a sound is loaded the first time.

Added `ANIM_ONCE` spawnflag to `model_spawn`. If set, `model_spawn` will cycle through its animation sequence once, then toggle itself off. The animation sequence is repeated each time the `model_spawn` is triggered. What's this good for? For an example, see the lightning in the *rain* example map.

`Freeze` developer console command allows you to freeze all entities other than the player in place. This is the ancestor of what will become a new powerup in subsequent versions, and is also very useful for screenshots. Thanks **venomus** for the suggestion.

- Squashed Bugs -

Version 1.8 introduced a bug with `func_door` `movewith` children. The child would only move to its new location after the door had completed its move. Thanks SoftShoulder.

Ricebug correctly pointed out that the `movewith` feature didn't work properly with teleporting trains. Fixed it.

Jed pointed out a problem with `func_tracktrain` that would occasionally cause the train to flipflop 180 degrees. This should no longer be possible.

A couple of problems with `target_playback` are fixed:

- In version 1.8, a 3D `target_playback` that "movewith"ed a moving entity but was **not** currently playing would foul up the volume/location of any other 3D `target_playbacks` in the map. Of course this wasn't a problem if the map only contained a single 3D `target_playback`.
- Version 1.8 wanted to play `target_playback` sounds so badly that it would unceremoniously crash the game if `fmod.dll` was not present.

April 12, 2001

Version 1.8

- New Features -

Choo-choo! Half-Life-like `func_tracktrain`, `func_trackchange`, and `path_track`.

New spline pathing for `func_train` and `model_train`.

Added 3D, MUSIC, TOGGLE, and START_ON spawnflags to `target_playback`. Of these, 3D is the most significant, but you'll also want to check out MUSIC. What this does is allow you to distinguish between background music that the player may not care to listen to and more important voiceovers or other sounds. Also added fadein and fadeout values, which allow you

to increase/decrease the volume of a sound over time. This version also supports music files (.MOD, .S3M, .XM).

Related to 3D target_playbacks, added "attenuation" and "shift" values to **worldspawn**. "Attenuation" controls the rate at which the volume of a sound decreases with increasing distance, and "shift" controls the Doppler shift effect.

Func_pendulum is just what it sounds like. Check out the pendulum map in the **examples**.

Added custom sounds to **item_jetpack**, provided by Jeff Hayat (*jeffrey*). Check 'em out in the rottrain example map. Jetpack is now selectable from the inventory.

Func_explosive now has a delay value that can be used to trigger a chain reaction of explosives without annoying SZ_GetSpace: Overflow errors. Thanks Ricebug.

Target_anger has been changed a bit so that it will no longer automatically pick the first match on its killtarget value. Instead it will select the entity that's closest to the angry dude and that has health > 0. So if you're looking to build a scripted sequence in which one or several monsters concentrates on killing an army of noncombatants (misc_insane, for example), well... you can do that now.

The code now takes fog density into account when figuring visibility. If visibility is less than 5%, a monster can no longer see his enemy. If visibility is less than 12%, the monster's aiming accuracy is decreased.

"Distance" value added to all monsters. An enemy farther away than "distance" is effectively invisible to a monster. In standard Q2 this distance is fixed at 1000 units; we've made the default a slightly meaner 1280 units. Now if you want that gladiator to rail you when you're 2000 units away instead of standing around like a slacker, well he will.

We've listened to feedback from players who aren't big fans of our perfect-aim Gunner. The first cut at reducing gunner grenade accuracy was to add a random component to his aim on easy and normal skills. However, since the guy fires 4 grenades in rapid succession the end result was to actually make him tougher, depending on how you normally play. In this version the gunner keeps the same perfect grenade aiming on all skill levels, but will only fire the 1st and 3rd grenades on easy skill, and skips the 4th grenade on normal. Your opinions are welcome.

Model_train can now be animated in the same way that **model_spawn** is.

A couple of new **developer** console commands: "spawn *classname*" adds any point entity to a map, facing the same direction as the player and 64 units in front of the player. Look slightly up to prevent embedding spawned monsters in the floor. The entity can then be moved around using the normal Lazarus **item movement commands**, and position info can be retrieved using the **id** command. Of course in the case of monsters you'll want to do this with *notarget* set. Also added "spawnself", which drops a copy of the player model in the map. This might be useful for screenshots.

- Squashed Bugs -

Fixed a problem with NO_GIB monsters. The intent all along has been that if dead NO_GIB monsters blocked the path of a moving brush model (door, train, etc.) then those monsters would explode. The implementation was off, however. This version works.

In previous versions an integer overflow problem existed with the jetpack that would cause you to be propelled very quickly to the vertical extents of the map if you had been playing the map for a while (but not using jetpack).

Previous version contained a game-crashing bug on Windows 2000 that occurred if you saved and reloaded a game while using **third-person perspective**. Thanks Nightmare66.

February 26, 2001
Version 1.7.1

Restored solidstate key for **model_spawn** to its original meaning. Solidstate=1 is nonsolid, not affected by gravity. New solidstates 3=solid, not affected by gravity and 4=nonsolid, affected by gravity.

Modified player physics just a bit from 1.7. In this version, prediction is not turned off for a player riding a brush model unless that brush model has "turn_rider" set. In version 1.7 prediction was needlessly turned off when riding **any** brush model, which resulted in a bit of a laggy feel to the game.

February 23, 2001
Version 1.7

- New Features -

Added LOOK_TARGET spawnflag to **trigger_look**. This form of trigger_look requires the player to be both inside the field of the trigger, and to be looking at the targeted entity, before that entity will be triggered. Multiple entities with the same targetname may be targeted by the same trigger_look, and only the one that's being looked at will be activated.

Added **target_playback**, which makes use of FMOD by **FireLight Multimedia** for playback of alternative audio formats, including .wav, .mid, and .mp3. A **BIG THANKS** to Firelight Multimedia for producing this product to start with, and then allowing us to redistribute their dll. This is cool.

Added "sound_restart" console command as an alternative form of the "snd_restart" console command, to avoid sound oddities when restarting the sound system when FMOD.DLL is loaded.

Added "sv_maxgibs" cvar (default=20), which specifies the maximum number of gibbs that may be spawned in any 0.1 second frame. This feature will help eliminate many occurrences of SZ_GetSpace Overflow game pauses.

Added "hint_test" console command. With developer=1, look at any monster and type "hint_test" at the console. The monster will ignore normal AI and seek a hint_path, then continuously run back and forth along the corresponding hint_path chain. To return the poor guy's brain, look at him again and type "hint_test" at the console once more.

Changed "hud" console command to optionally take a state argument. "hud" toggles the hud as before, "hud 0" turns the hud off, "hud 1" turns the hud on.

Added "texture" console command to report texture name, surface flags, and value of the brush viewed. Developer must be set to 1 to use.

Added "style" key for info_player_start, info_player_deathmatch, and info_player_coop. If non-zero, this value specifies the player's starting weapon in the same way the worldspawn style key does, and overrides the worldspawn setting, if any. For map transitions, a non-zero style value for info_player_start will remove all weapons and ammo from the player and only give the weapon (and ammo for that weapon) specified by style.

In most cases the player should now be able to ride pitching and/or rolling brush models without being eaten by the model. This problem has always been with Q2 and is likely a big reason you don't see many rideable rotating entities. For flat platforms the code is fairly foolproof - you should **never** take damage from the brush you're standing on. There is, however, still a problem with vertical components of brush models. Lazarus currently pushes a player up until his bounding box won't intersect any of the model's brushes... which beats being eaten by the model but may result in the player being tossed off of the model, particularly if it is translating as well as rotating.

Added REVOLVING spawnflag (=2) to **func_door_swinging**. If set, the door is always "closed" and will rotate in the same direction when opened or used again. However, the direction of rotation is now determined according to which side of the origin the activator (or damage for shootable doors) is on, rather than simply being tied to which side of the door the activator is on. This feature will come in handy when used in conjunction with "the next big thing". Stay tuned.

Func_pushable with health<0 option makes it truly indestructible, and will block moving brush models. It will likely take a bit of trial and error to get the brush model speed adjusted such that it will appear to touch the func_pushable when blocked. The trick is to set the speed such that (travel distance minus blocker dimension + 2) divided by the speed is exactly equal to or slightly less than an integer multiple of 0.1 seconds. For example, a 128 unit tall door with lip=8 that moves vertically will come to rest on a 64 unit tall func_pushable with speed=97 (0.598 seconds) or 116 (0.5 seconds).

Improved **hint_path** code; it's now much more effective. The original code was designed to find the hint_path closest to a monster's enemy as the monster's goal, but... didn't. This didn't cause any problems, but **did** cause monster to restart the hint_path sequencing code more often than was necessary.

Several changes to medic AI (these guys are really annoying now):

- The most significant change is that when leaving his idle state (**only** after being awakened by being triggered or angered in some other way) the medic will now look for a hint_path entity, and if found, follow the corresponding hint_path chain looking for dead monsters to resurrect. To facilitate this a bit, if medic has not seen or heard his enemy in more than 20 seconds he'll return to his idle state, which means he'll potentially give up on finding the player and go follow hint_paths instead. Just as angered monsters never give up looking for the player, such a medic will never give up looking for buddies to resurrect.

Medic recognizes transparent brushes (like windows) are solid, and will not dumbly attempt to heal a monster on the other side.

- If medic cannot reach his intended patient in 5 seconds (e.g. if the patient is on the opposite side of a lava pool), he'll now look for a hint_path that leads to him. If found he follows it.
- Medic will now give up attempting to heal a monster after 10 seconds (if, for example, the monster is more than 400 units away and the medic cannot find a path to the monster). Medic will not attempt to heal that monster again for at least 60 seconds.
- Added a check to ensure that a clear line-of-sight exists between medic "muzzle" and the dead monster before initiating cable attack. This is **not** a guarantee of success, since medic animations cause the cable origin to move around a bit, but it should prevent a few occurrences of medic attempting to heal a monster when the cable can't get there.
- Added "medic_test" console command. When turned on, medic will begin moving to hint_paths **without** waiting to be angered. This feature is of course best used in conjunction with "notarget" to see the medic's fancy new moves.
- Healed monsters now display the red shell for the length of the medic's cable attack animation, which seems to be the intent of the original code.

Added ability to use custom skins (along with the original skins) for all **monsters**. The skins are up to you, though... artists we definitely are not. This feature makes use of Argh!'s player model transmogriifier code.

Target_bmodel_spawner is a new point entity that clones existing brush models. The intention here is to help prevent ERROR:Index overflow.

Added CHASE_CAM spawnflag (=1) to **target_monitor**. If set, camera will follow the target_monitor's target by "distance" units, raised "height" units above target's origin. Default distance=128 units. Note that "movewith" is incompatible with CHASE_CAM (if both are used, CHASE_CAM is turned off).

TRACK Turrets are **much** deadlier now. In previous versions turret would set its angles based on the target in the **previous** (rather than the **current**) frame. Turret AI is tied to skill level. Skill 0 is the same as standard Q2: turret finds desired angles but does not apply the corresponding changes to angular velocity until the following frame. On skill 1 those changes in angular velocity are immediate. Skill 2 and 3 are the same as skill 1, except that the "target" is a prediction of the target's position 0.1 seconds in the future.

For **target_laser**, "mass" value now specifies the laser diameter and overrides the FAT spawnflag. Mass must be greater than 1. If mass=0 or is left blank, the default diameters will be used (4 for normal, 16 for FAT). "Style" value may be used to 1) create a no damage laser (style=1,2, or 3), 2) create a laser that monsters will ignore (style=2 or 3), or 3) create a laser that doesn't produce sparks at the receiving end (style=3).

Added ability to use a **target_anger** to make monsters attack a brush model. Brush models used for this purpose **must** have an origin brush.

Added ENVIRONMENT spawnflag (=64) to **trigger_hurt**. If set, players using the environment suit will not be damaged by this trigger_hurt.

Changed GOOD_GUY behavior slightly in that if a player triggers a GOOD_GUY, that flag is turned off and the monster attacks the player as a normal monster would. This change avoids a bit of silliness in that used GOOD_GUY monsters will no longer attack the player, only to then come to the player's defense if the player is attacked by someone else.

- Squashed Bugs -

Fixed a horrible bug that has existed in the last several public releases. The problem was associated with saving games. As part of an optimization frenzy several versions ago, saving a game caused "player_noise" entities to be removed. This was done to prevent duplicate player_noises from spawning into a map that you had left and returned to. The change worked fine for the intended purpose, but was A VERY BAD THING when saving a game in the middle of a map and continuing to play. This bug may have caused newly spawned entities to appear at the player's location when he jumped or fired his weapon, or may have caused unexplained crashes when alerting unseen monsters.

Solved a problem (we hope) with switching from software or "Default OpenGL" to "3dfx" in maps with fog. The problem with changing video while fog is active is that the gl_driver setting is updated before the actual vid_restart, so that the code would send Glide instructions to your card before it was ready. This resulted in a hardware crash with Voodoo 3/4/5 requiring a cold restart and a slightly less obnoxious game crash with a Voodoo 2. With this version, the code preserves the frame number when using software rendering or when OpenGL or Glide fog is used, and will NOT use 3dfx fog calls until at least 1 second has passed since the last time the code detected software or OpenGL. When switching to 3dfx you may

notice a short hiccup with no fog, followed by the correct fog. Why does this matter? Well if you switch between rendering dll's with a Voodoo, in order to use `gl_showtris`, and the map has fog, it won't go boom no more.

Fixed a problem with collision detection for brush models that do **not** use an origin brush. In previous versions, in some circumstances a `func_pushable` would not block the path of a closing door or moving train when it should have.

In previous versions if a `trigger_fog` was active when a game was saved, and that game was later loaded, the fog parameters were a bit unpredictable.

Fixed a fix. Several versions ago we fixed a bug in standard Q2 that resulted in rotating bmodels that were nowhere near dead monsters gibbing those dead monsters when the player would walk through them. Unfortunately this fix resulted in ALL moving brush models now simply passing through dead monsters. Fixed it, happily w/o breaking previous fix.

A few long-standing problems with medics are fixed.

- The first problem has been around since the original game. When medic heals a monster, the healed monster's enemy is set to the medic's oldenemy, if oldenemy is a player. If medic's oldenemy is NOT a player (for example it's another dead monster), then the recently healed monster is completely oblivious. Get in his face and he just smiles until you jump or fire. This problem was due to the monster preserving his enemy - he still was mad at you, he just didn't know it :-). So now... if medic doesn't set a healed monster's enemy, the enemy is set to NULL and the monster will find an enemy through normal AI stuff fairly quickly.
- Another original game bug was the occasional persistence of the red shell on revived monsters. Up to now the code has depended on the medic, not the healed monster, to turn the shell effect off. Well it looks to me like it should work, but still sometimes does not. Short version is the code no longer depends on medic to turn the effect off, but on the monster to do so.
- A Lazarus problem with the medic which arose from making one change without beefing up something else is now also fixed: Lazarus medic prioritizes targets differently than the standard game - if a dead monster is within 1000 units, he'll always attempt to revive that monster rather than attack the player. This has been in place for a couple of versions. Problem was if the medic was within 1000 units of dead monster but farther away than the max healing range (400 units) he'd take a couple of baby steps toward the dead monster every 3 or 4 seconds then stop. This was particularly noticeable if you used "notarget". In this version under the same circumstances medic will run to within the max healing range of the dead monster.
- Also, code now checks for other monsters or the player being entangled with a dead monster. If another entity intersects a dead monster's bounding box, the medic will not attempt to revive that monster.

`Misc_actor` with a `combattarget` was a moron. Recent "ideal range" code made it impossible for the actor to decide between running to a `point_combat` and running to his enemy. Thanks Jedi for the bug report.

Fixed a minor annoyance with `turret_breach`. Depending on `turret_breach` and `turret_base` construction, in previous versions when operating a player-controlled turret and looking down, the game would play footstep sounds as though the player were running downhill.

Also for `turret_breach`, fixed a problem with grenade-firing turrets with low grenade velocities (distance value). Previously the grenade aiming calculations tended to get stuck in an endless loop searching for a good trajectory. Also eliminated an annoying jitter in grenade-firing turrets that would occasionally show up when the player was at a very close range.

Fixed a mistake in initial weapon selection code (`worldspawn` "style" key). Previously if `style<0`, no ammo was given for the default weapon.

Fixed a zoom-related problem with `joy_cvars`. Previously these cvars were always forced to default values at startup.

Both `trigger_fog` and `target_fog` were fouled up (but in different ways) if both a delay and a 180 deg. density were used. Thanks Argh!

Fixed a problem associated with OWNED_TURRET `trigger_look`. In previous versions player position was not updated as the turret turned. The view was correct, but the invisible non-solid "player" may or may not have been in the correct position to trigger the `trigger_look`.

January 18, 2001
Version 1.6

- New Features -

Added **func_door_swinging**, a rotating door that always opens away from its activator.

Added dmg key to **func_force_wall**. If non-zero, touching an active **func_force_wall**... hurts.

Func_vehicle driver may now look around while driving. Driver still turns with the vehicle. **Func_vehicle** can act as a **movewith** parent. Added **turn_rider** key to **func_vehicle**, which effects any **movewith** children (but not the driver).

Func_water and **func_bobbingwater** have a new MUD spawnflag. Mud slows player progress considerably and will pull the player in if deep enough.

"hud" console command toggles the display of the HUD. Useful for demos and screenshots.

Added **item_jetpack**. Fly around your map.

Misc_actor:

- New player model transmogrification code from **Argh!** now makes it possible to use id male, female, and/or cyborg player models without redistributing those models. This is very cool.
- Modified accuracy of actor firing, which previously was too sloppy.
- Added "run away and hide" code for actors with low health. This feature can be disabled by setting the **actorchicken** cvar to 0. Chicken actors find new courage if their enemy is attacked by someone else.
- Actors now seek cover immediately after completing their attack sequence. Actor will pause in his hiding spot from 0-6 seconds before resuming attack. This feature can be turned off with the **actorscram** cvar.
- Added quite a bit of smarts to actor decision-making concerning his preferred range to target. You should no longer encounter protracted battles between two face-to-face actors.
- Rocket-firing actors fire at the target's feet roughly half the time. For all other weapons actors fire at the target origin. In both cases, actor first checks to see that the desired target is visible.

Model_train, **misc_viper**, and **misc_strogg_ship** are now all capable of using the rotating abilities of **func_train**. **Misc_viper** and **misc_strogg_ship** can now be damaged by weapon fire.

Monsters:

- **Monster_soldier_X** accuracy is now a function of skill level. Easy=same as standard Q2. Nightmare mode gives perfect aim.
- Monster-fired rocket splash damage radius is now skill-level dependent, making tougher skill levels quite a bit deadlier.
- If enemy is not visible at end of **monster_chick**'s windup phase, she'll now fire at the feet of enemy's last known position.
- **Monster_soldier_light** blaster bolt speed is now skill level-dependent, = $600 + \text{skill} * 100$.
Monster_chick now goes through her windup sequence a bit faster, depending on skill level. Out of the 13 frames, 2 frames are skipped on normal, 5 frames on hard and nightmare.

Added **target_change**, which can be used to alter characteristics of other entities. For example, you can change a **path_corner**'s target to a different **path_corner**, so that a **func_train** follows a different path.

Added NOGUN spawnflag to **target_changelevel**. If set, on the next map or demo the cvars **crosshair** and **cl_gun** will be 0. These get reset to previous values when exiting that next map or demo.

New **target_fade** allows you to fade the screen to a specified color and opacity.

Added **target_failure**, a mission-ending event used to inform the player that... he messed up. In conjunction with this, added **NO_BACKGROUND** spawnflag to **target_text**.

Target_movewith can be used to tie the motion of one entity to another, or remove that association.

Target_set_effect is used to apply special effects to point entities.

Target_sky allows you to change environment maps during the game.

Added Rogue's **trigger_disguise**, which effectively makes the player invisible to monsters until the player fires a weapon or another **trigger_disguise** with the **REMOVE** spawnflag is touched.

Added NOGIB spawnflag to **trigger_hurt**. Changed trigger_hurt code to check for the presence of a damageable entity at activation. In standard Q2, if a monster is standing inside a trigger_hurt when it is activated, he won't "touch" the trigger until he moves or reacts to the player.

Added SILENT spawnflag to **trigger_key**. If set, neither the success or failure sounds are played, and the "You need the ..." message is not displayed. This is useful for removing items from players.

Trigger_transition may now be used with LANDMARK **trigger_teleporter** and **misc_teleporter**.

Changed **turret_breach** such that it will rotate with a "turn_rider" movewith parent **unless** it is under the control of a player/monster. Turret_breach can now fire grenades and hyperblaster bolts.

- Squashed Bugs -

Fixed a standard Q2 bug that caused rotating brush models to gib a dead monster even though the monster was **nowhere near** the rotating brush. This would usually occur when the player would walk through the dead monster.

Changed monster jumping code so that monsters/actors will NOT attempt to jump if their path is blocked by another monster, actor, or the player. The code has always prevented monsters from intentionally jumping onto other monsters or the player, but monsters were still very prone to attempt to jump **over** crouched monsters. While entertaining, this was a bit goofy. One unexpected jump side effect that we'll likely leave in... it takes a bit of patience, and circumstances have to be just right, but if you fire rockets at fast-moving monsters like the berserker you'll see it sooner or later.

Solved a long-running feud with the Q2 physics that now allows you to ride a turn_rider rotating platform without an exceptionally jerky ride. Take a ride in the helicopter in the lcraft example map.

Fixed a game-crashing bug that occurred when shooting GOOD_GUY **misc_actors**.

Func_vehicle now works correctly if given an initial yaw angle. Also, func_vehicle will now plow through multiple damageable entities rather than being stopped cold by the first obstacle.

Added a workaround to make misc_blackhole visible. This is a strange error shared by the Rogue MP... no idea what's going on here, but hey, it works.

Target_changelevel with CLEAR_INVENTORY spawnflag now properly restores player health to 100 and forces a weapon change to the blaster (or no weapon, depending on **worldspawn** style key).

Fixed a problem with **target_spawner** due to changes associated with **trigger_transition**. Gibbed monsters had their classnames changed to "gibhead" to facilitate map transitions. This was fine, but the target_spawner code formerly used the same address for the spawner's target field and the spawned entity's classname. This resulted in correctly spawning a monster the first time, but if/when the monster was gibbed then subsequent target_spawner uses would spawn a gib head rather than a monster.

Fixed a game-crashing problem with dmgteam monsters/actors. Previously if a dmgteam member was alerted to come to his pal's defense but could not see the attacker, under some circumstances the game would suddenly crash to the desktop.

Fixed a problem with **model_spawn** that prevented a NO_MODEL model_spawn from being toggled. Thanks SoftShoulder.

Fixed another problem with model_spawn (hopefully permanently this time) that would cause a solid model_spawn to crawl up your leg when standing on top of it.

- Other Changes -

Added a feature to help track/prevent Index Overflow errors. If readout=1, dll displays model and sound indices as they are used. Readout MUST be set on the command line or in a .cfg file; once the game starts it is too late. For best results start the game with "+set logfile 2 +set readout 1". After walking through the map, go to a DOS prompt in the game folder and type "sort qconsole.log >index.txt" (or whatever name you like) to make the list a bit easier to read.

November 3, 2000
Version 1.5.1

- New Features -

Several **monsters** and **misc_actor** can now jump up or down to reach their intended target, with no intervention from the mapper. Berserkers are much happier now.

Monster AI - not-so-dopey-after-all monsters won't run into lasers or run closer than their present position to live grenades.

Added **IGNORE_FIRE** spawnflag to **monsters** and **actors**, which basically tells other monsters/actors to ignore friendly fire from this guy. This will keep monsters focused on what they do best - killing marines rather than fighting among themselves.

ANY non-flying, non-exploding **monster** may now have flies at death. Or even before, heh. We've also fixed a bug in standard Q2 mutant and infantry code, which caused those monsters to ALWAYS display flies when the original intent was to have flies only half the time.

Expanded **misc_actor** **GOOD_GUY** behavior to give those guys a bit more freedom in killing monsters. After knocking off available enemies, **GOOD_GUYS** will then follow the player around to provide further assistance. Also added **+use** feature to tell **GOOD_GUY** actors to get out of the way.

If bounding box coordinates or muzzle offset locations for **misc_actor** are omitted, default values for **bleft**, **tright**, **muzzle**, and **muzzle2** shown in the **ActorPak** documentation will be used. This should be a nice time saver, assuming you use an **ActorPak** model and you agree with our defaults.

Misc_viper_bomb can be used multiple times and can specify the entity that "drops" it, rather than always using the first **misc_viper** in the map.

Fog. **Trigger_fog** can now start off, be toggled, and have a delay value. Both **trigger_fog** and **target_fog** can change density according to the player's viewing direction.

Path_corner wait < 0 now properly causes monsters/actors to wait at the **path_corner** until another external event causes them to move.

Added **target_monitor**, a viewing station that can be used to show remote events to the player.

Added **target_animation**, which can be used to force actors and other entities to play a specific animation sequence. This might be useful for scripted sequences, and can also be used to give actors a bit more personality.

Added **trigger_teleporter**, a brush model teleporter that can be any size.

Trigger_bbox can now be made shootable (while remaining non-solid).

Func_door can now act as a **movewith** parent.

Added **trigger_transition**, a trigger field that allows monsters, weapons, and other point entities to change levels along with the player due to a **target_changelevel** activation.

Trigger_hurt and **target_laser** heal the touching entity if **dmg** < 0 without the annoying damage effects (blood, "Arghhhh!!"). Also, if **dmg**<0 and **SILENT** spawnflag isn't set, **trigger_hurt** plays the **item_health_small** pickup sound rather than **electro.wav**.

- Interim release new features -

You can now specify which weapon the player spawns into the game with (or choose no weapon at all). Please note that this feature is strictly experimental at this point, and we do not recommend using it in released maps yet. Choice is made with the **worldspawn** "style" key, -1=no weapon. As of now, there's no way to force unknown player models to use the "no weapon" **w_null.md2** for a weapon model, so in DM or if using thirdperson view, unless the player is using the male model he will appear to be carrying his model's default weapon (**weapon.md2**) even though he really doesn't have any weapon at all.

Added **weapon_blaster**, which really only makes sense to include if the player starts with no weapon.

Added **LANDMARK** spawnflag (=64) to **misc_teleporter** and **trigger_teleporter**. If set, the user's angles and velocity are preserved, rotated if necessary by the destination angles minus the teleporter angles.

- Squashed bugs -

Path_corner now distinguishes between **path_corner** and **target_actor** for its next target, so that **target_actor JUMP spawnflag** doesn't cause a teleport.

Fixed problem with **target_anger** that prevented pathing function from working unless a **killtarget** was also specified.

Fixed a standard Q2 bug with **trigger_hurt** that would prevent normal sound from being played 90% of the time if the "slow" spawnflag was set.

- Interim release bug fixes -

Fixed a nasty bug affecting 3dfx users only. Previously if a 3dfx user entered the game using software rendering, the code inappropriately turned fog off using a Glide call, which... was not good.

Trigger_transition:

- Save/Load games didn't work at all, or putting a good face on it, worked too well. Entities which had changed levels were duplicated - one in the spot where the game was saved, one in the location where the map first started up.
- Monsters in the middle of a death animation at the time a **target_changelevel** occurred were stuck in that frame until returning to the previous map.
- After switching levels then switching back, **misc_actor** lost his weapon.
- Dead actors who changed levels regained a stationary weapon along with a stream of **R_CullAliasModel** errors.
- Code now correctly accounts for users running Lazarus from baseq2 folder when writing/reading entity **changelevel** file.

Misc_actors temporarily lost their minds after switching weapons, and would not fire until fired upon.

If a player acquired a new weapon immediately prior to changing levels, so that the new weapon didn't have time to come up, you'd be flooded with a stream of **R_CullAliasModel** errors in the new map.

In previous version if you triggered a "turn_off" **target_fog** with a delay before any other fogs were activated, a solid black fog faded away. In this version the same situation results in... nothing, which is correct.

Not so much a bug, but an improvement: fake player replacement for player using a **target_monitor** or **func_monitor** is now animated using the player's stand animation, rather than being frozen in place.

October 1, 2000

Version 1.4

- New Features -

Added **misc_actor**, a combatant with a choice of models and weapons that can be a good guy or a bad guy.

Added Rogue's **hint_path** entity, a device intended to assist monsters in tracking down players, thereby making monsters appear quite a bit smarter than in the standard game.

Added LANDMARK spawnflag to **target_changelevel**, allowing you to have HL-like transition zones from one map to the next.

Added **target_cd**, which allows you to specify a CD track to play a specified number of times.

Added OWNED_TURRET spawnflag to all trigger field entities (**trigger_once**, **trigger_multiple**, **trigger_look**, etc.) This allows a player who is currently viewing the map through a **turret_breach** camera to "touch" a trigger.

Target_anger now includes the scripting capabilities of **target_actor**.

Several features added to help make the game run more efficiently and prevent "ED_Alloc: No free edicts" errors:

1. **count** key/value added to many entities, allowing you to easily remove entities from a map after being used a specified number of times.
2. **Target_locator** now correctly removes itself after performing its function at startup.
3. **Trigger_key** without the Multi-use spawnflag is removed once used.
4. Single player game no longer reserves 8 slots for "bodyque" entities, which are **only** used in deathmatch and coop games.

Added FLIES spawnflag to `misc_deadsoldier`.

- Squashed Bugs -

Sitting, decapitated `misc_deadsoldiers` exploded when gibbed in previous versions. This was due to a conflict between `misc_deadsoldier` spawnflags and `NO_GIB` spawnflags for monsters.

Fixed a bug in `target_text` that resulted in a crash if a line of text consisted of 35 or more characters with no spaces. Also added support for `\n` line breaks and fixed a number of inconsistencies with word-wrapping and formatting codes.

Fixed a game-crashing bug with `target_anger`. Previous versions would cause a non-combatant killtarget to crash the game when "reacting" to damage.

Solved a game-crashing bug that would sometimes occur if using `third-person perspective` during a level change.

September 4, 2000

Version 1.3

- New Features -

Added `target_attractor`, a sort of entity magnet with many potential uses.

Added `trigger_look`, a special-purpose `trigger_multiple` that activates its targets only when the player is looking at a specific area.

Added `trigger_bbox`, a point entity that allows trigger field functionality without a brush model.

Improved performance of the `movewith` feature of several entities, especially `func_door` and `func_button`. Also added support for many new entities.

Added `console commands` that allow a mapper to move entities to the desired location in a map. Mapper can then use the `id` console command to get the entity's position and other information.

`Func_pushable` can now be dropped on monsters or players, resulting in damage and/or death to the squashee.

Floating `func_pushable` now takes a rider's weight into account. 64x64x64 crates will generally sink about 6 units with a player rider; 32x32x32 crates will generally sink to the bottom.

Added recursive brush model-rider movement code. In other words, a player standing on a `func_pushable` box on a box on a box on a moving `func_train` will move along with the `func_train` (as do the `func_pushables`, of course).

Added custom sound spawnflag to `trigger_push`, so that you are no longer limited to using the default wind sound.

`Target_locator` now copies the `angles` value of the `path_corner` to the moved entity (for point entities only, not brush models).

- Squashed Bugs -

Corrected a problem with `func_pushables` riding a rotating brush model. Note that if the brush model has `turn_rider` set, then the `func_pushable` **must** use an origin brush.

Solved a map order bug related to `target_lasers` with `movewith` `func_trains`.

In previous versions, monsters placed in the map with a `target_locator` lost their minds and basically became statues. Fixed it.

- Other Changes -

`Func_pushable` bounding box has been changed slightly. This change is for the better, but **may** require you to re-work old maps if you've used stacked `func_pushables`.

`id` and `bbox` developer console commands now find pickup items as well as solid brush models and point entities.

Decreased height of monster_gladiator bounding box by 16 units to more accurately reflect model dimensions. This was done mainly for the benefit of dropping things on the gladiator, but might also now mean that a gladiator will have access to map areas that he previously could not enter. All other monster dimensions are more or less correct, with the exception of the floaters. We've all seen floaters bury their heads in the ceiling. The only way to correct this problem is to increase the size of the bounding box, but this change might result in breaking existing maps.

July 30, 2000
Version 1.2

- New Features -

Added **movewith** key to many entities. If used, the entity's origin, velocity, and rotational speed is relative to its "movewith" func_train or model_train. Future versions will most likely allow additional entities to serve as *movewith* parents.

Added SMOOTH_MOVE spawnflag to **func_train**.

Added **turn_rider** value to all rotating brush models. If non-zero, players standing on that brush model will rotate with the model. Also added a turn_rider server cvar, so you'll be able to turn this feature off for network games (where prediction might foul things up a bit) or if you just don't like turning along with rotating objects.

Added **target_effect**, a souped-up target_temp_entity.

Added **func_force_wall**, a force field with a new particle effect from the Rogue Mission Pack.

Minor changes to target_rocks: It is now much less likely that a rock will get hung up on the side of a cliff and spin in place. Rocks now use the same 2-second fade that Lazarus gibbs use, rather than going "poof!"

Fixed a problem in original Q2 monster AI - trigger spawned monsters could not target a path_corner.

Added noise fields to **model_spawn** and func_rotating so that these entities will play a sound when active.

Player velocity matches the entity he just jumped from, so, for example, jumping up won't cause the func_train you were standing on to run out from under you.

Player velocity is added to initial grenade launcher grenade velocity. So if you're running after the helicopter in the example maps trying to lob a grenade through the door, you might actually succeed :-). We're interested in your opinion on this change, so if you have a gripe please let us know.

Changed NO_GIB **monster** damage effect from TE_SCREEN_SPARKS to TE_SPARKS.

- Squashed Bugs -

Fixed a bug in **tremor_trigger_multiple**. After toggling off, then on, the trigger reverted to normal trigger_multiple behavior.

Ack! Fixed a bug introduced when fixing a Q2 bug. Previously, a path_corner w/o a target that was targeted by a monster would crash the game, even if wait was set to -1.

July 18, 2000

- New Features -

Version now bumped to 1.1.

Added rotating **func_train**. Big thanks to Rroffstein (Martin Painter), who provided the basis for this code. See the whizbang **rottrain.bsp** in the **examples**.

Fixed several annoying bugs in the original Q2 source concerning **monster** movement 1) away from path_corners with a wait value and 2) to initial point_combats.

Added CUSTOM and LOOP spawnflags to **target_lightramp**. These changes allow you to use any pattern you want when switching lights on/off. This also effectively allows switched flashing lights, which isn't possible in standard Quake2. See **lramp.bsp** in the **examples**.

More smart monster stuff: Monsters will now run away (and possibly hide) when damaged by anything that they don't normally get mad at - closing doors, trains, lava, target_blasters, etc. This will continue to improve. An example of this new behavior is shown in **monsters.bsp** in the [examples](#).

- Squashed bugs -

Fixed several problems with turret_breach (it's perfect now :-)). Under some circumstances blocking the movement of a TRACK turret_breach would cause a game crash. Also (this was really annoying) normal [turret_drivers](#) were animated along with the turret_breach they were operating, resulting in some very peculiar dance moves.

Fixed a problem with [fog](#) for normal GL cards. In previous version the fog was **extremely** dense on these cards, even though it appeared more or less right using the Mesa GL drivers on a 3dfx card.

Fixed a problem with fog being retained across a target_changelevel or gamemap command. Fog will now be turned off when switching maps with either the gamemap command or a target_changelevel (this was always true with the map command). One possible **source of confusion** here: if you stay within the same unit and leave a map that has fog on and then return to that same map, fog will be on whether the map you just left used fog or not.

Ack! If you downloaded lazarus.zip since Friday July 14, you've undoubtedly noticed that the rottrain example immediately crashed. This wasn't a code problem but a pak-editor-ate-my-textures problem. This has been fixed.

In previous versions if the player managed to jump on top of a solid model_spawn, the model_spawn would... umm... crawl up the player's leg.

In previous version if the player exited a trigger_fog while a "turn off" target_fog with a delay was still ramping down, results were... not as expected. This works correctly now.

- Other Changes -

Player must now use +moveup (jump) to exit a player-controlled [turret_breach](#). In previous version, +back was used. However, for users who don't normally use freelook this made the turret exceptionally difficult to control.

Player riding on a floating func_pushable is now transported with the func_pushable. More importantly, we've eliminated the annoying jitter when riding a func_pushable.

July 6, 2000

Umm... changed our name, and welcome to our new home. Concurrent with this event, version numbering is begun with 1.0.

Several minor bug fixes and one major PC-crashing bug fix to the [turret_breach](#). Previously, if you were using a func_monitor to look through a turret_breach and the map was changed due to a target_changelevel or console gamemap command... well let's just say the result was not good.

Added *followtarget* key to turret_breach. If no player or monster targets are found, turret will follow (not fire at) the entity specified by this value.

Added machinegun fire to turret_breach.

Several procedural changes and one visual glitch fix to [monster_medic](#).

A couple of transparent changes to [zooming](#) make the thing operate a bit more logically.

June 28, 2000

Changed a few things up to help prevent borking your m_pitch setting when using zoom; also removed the necessity of using dhpak.cfg.

Minor changes:

- In previous version, if you had *notarget* set and switched to third-person view, the fake player did **not** have notarget set and was fair game for monsters and automated turrets.
- Zoom is automatically turned off when you switch to third-person view.

- If you are zoomed in while controlling a turret_breach via a func_monitor, the turret_breach will not grab control back from you. Normally, an automated turret takes over control if the viewer does not change the view angles for 5 seconds.
- Added a fake crosshair to the zoomed view. Think you can do better than \pics\zoom.pcx? I bet you're right... how 'bout sending me a pic?
- In zoomed view, "muzzle" is moved to the player's viewpoint, rather than being placed at the normal 8 units below the view. If you place the crosshair on a target with a hitscan weapon... you **will** hit it.

June 27, 2000

As promised, with this version we have an example map illustrating some of the very cool things you can do with **target_rotation**. See rotation.map in the example map files.

Still more changes to **turret_breach**:

- You can set a spawnflag on **turret_driver** to make him operate the turret_breach remotely.
- Turret_breach is now animated when "owned", either by a turret_driver, a player driver, or a player viewing the turret_breach with a **func_monitor**. What's this useful for? How about a pulsing light to indicate when the turret_breach is being activated by another player/monster (as opposed to an automated turret)?

New continuous **zoom** feature that adjusts mouse and joystick sensitivity. This is pretty slick, but only works for single-player games and requires you to use a custom cfg (distributed with DHPak).

Changed code so that trigger spawned monsters are not counted in the total monster count until they actually spawn into the game.

June 22, 2000

A few more changes to the turret_breach/func_monitor combo:

- Player must be within 100 units of the func_monitor to use it.
- Automated or remotely controlled turret breach will not switch targets unless the new target is at least 100 units closer than the old target. This change helps prevent a pair of agitated berserkers from making you lose your lunch :-)
- Automated turret (and target_blaster and target_laser) will **not** acquire an invisible player (e.g. player embedded in a camera) and they **will** acquire the fake player taking the real player's place at the func_monitor.
- Added support for a "count" value for the func_monitor's target to specify the cycle order for cameras. If the count value is 0, the map order is used. This change will help the mapper correctly fill in the "viewmessage" value for the camera, e.g. "Camera #3".

Misc_teleporter adds 3 new spawnflags: START_OFF, TOGGLE, and NO_MODEL. It may also have multiple (up to 8) targets. The code currently selects a random destination from among the 8 or fewer entities that are targeted by the misc_teleporter.

June 21, 2000

Turret/monitor goodness:

- If the player is damaged while using the func_monitor, he's automatically switched out of camera mode.
- Multiple cameras may be accessed by func_monitor. Switch between cameras with the +moveleft and +moveright keys.
- Extreme trouble but worth it - the player will now see the correct muzzle flashes and hear weapon sounds from the camera's perspective. This part was no trouble, but then making monsters attack the fake player's body at the monitor was... interesting :-)
- Added "viewmessage" value for turret_breach. This message is displayed when a player uses a func_monitor for remote viewing.

Added **target_rotation**. This entity allows you to target an entity from a selection of entities, either in a sorted order or picked randomly. What's this useful for? Stay tuned.

Also fixed a problem with third person view: weapon change caused weapon to be displayed in the camera view.

June 19, 2000

Quite a few changes to **turret_breach**. You can create an automated turret ala Rogue, remotely control a turret with the new **func_monitor**, make a nifty security camera with a new "sounds" value, and create a popup turret using the new **TRIGGER_SPAWN** spawnflag. The **NO_DRIVER** spawnflag has been eliminated.

June 17, 2000

Fixed minyaw and maxyaw values for **turret_breach**. You may not have noticed before, but under some conditions these values didn't work properly, either in DHPak or the original game.

Added **NO_DRIVER** spawnflag to **turret_breach** for Rogue-like automated turrets.

Homing rockets now automatically blow up after flying for 8000 units. This is the same thing normal rockets do, only the homers die with a no-damage explosion rather than simply going *poof*. This change is designed as a failsafe for you guys with the quick feet who can manage to outmaneuver these things... it helps prevent game errors caused by having too many homing rockets flying around simultaneously.

June 16, 2000

Added lockon sound for homing rockets. Only the target of the homing rocket will hear the sound.

Trigger_relay may now include a message and sounds values.

Added **dmgtteam** value for **trigger_relay**. **Trigger_relay** will be triggered whenever a monster with a matching **dmgtteam** is injured.

Target_speaker - if attenuation is set to -2, only the player who activated the speaker will hear the sound.

... and please stay tuned. Mad Dog has hit me with a large collection of extremely cool things to do, coming this way soon.

June 13, 2000

Bah... my expert version control system works about the same as everything else I do. Basically it goes like this: "Well let's see how **THIS** works". Yesterday's distribution did **not** include homing-rocket firing monsters, despite documented promises to the contrary. Today's version does... I think :-)

June 12, 2000

You want tougher monsters? We bring you tougher monsters. :-) This is the first in a series of changes designed to make the monsters smarter, better opponents. In this version:

Gunners are **much** better grenadiers. Mad Dog suggested changing the gunner AI so that they would not fire grenades if the target was above them (which is what the Rogue MP does). But I thought... why stop there? Why not make them smarter, and have them **figure out** whether they can lob a grenade to a specific position and, if not, use the machinegun instead. So they do that. But much more importantly, if they **can** shoot you with a grenade, they do so much more accurately than either the original Q2 game (which was done really badly) or the Rogue Mission Pack, which was quite a bit better but still not correct. DHPak figures out the correct launch angle to get a grenade to the target. See the **monsters** section for a full description of the gunner's new AI. In case you were wondering, I'm pretty pumped up about this and other changes coming soon. See the last 2 rooms in **monsters.map** for an example of the new gunner AI. You'll notice that the gunners on the lower level won't fire grenades at you when you're on the top 3 levels - because grenades can't make it there. But step on down a bit and... heh... see for yourself. God mode is highly recommended :-)

Rogue source, with a few changes: Tanks and chicks do not fire suicidal rocket blasts into adjacent walls (which might happen with standard Q2 if the player strafes out of view). They will lead the target with rocket fire. The higher the skill level, the more likely they are to lead the target. If the target's feet are below the muzzle, then 2/3 of the time the tank/chick will fire at the target's feet. In all other cases they'll fire at the target's origin. Rocket speed is skill level dependent. On easy the default 500 units/sec is used. On nightmare it's a very speedy 800 units/sec. (Speed adjustments are not made if homing rockets are used).

Added **HOMING_ROCKETS** spawnflag to **boss2**, **chick**, **supertank**, **tank**, and **tank_commander**

Umm... whoops. Density calculations used for **func_pushables** were applied to **all** entities, and guess what? **Monster_flipper**

is too heavy to swim :-). This has been fixed.

Another whoops. Previously if fog was active when the user changed levels with the "map" command, the fog remained active in the next map. The code now correctly turns fog off at map exit.

June 10, 2000

Added health value to `func_train`, `model_train`, and `model_spawn`. IOW these entities may now be shot and destroyed. Removed the ineffective "block shots" solid state from `model_spawn` and `model_train`, which never worked.

Added `target_rocks`, used to generate a landslide

Added several features to `target_blaster`: It may now be toggled like a `target_laser`. It can target any entity, moving or not, including players. It has a choice of weapons.

`Target_laser` can target players. It can pulse on/off without the use of a `func_timer`.

Added `homing rockets` to available weapons for `target_blaster` and `turret_breach`. These things are nasty, the way you like 'em.

June 7, 2000

Added `target_anger` and `target_monsterbattle`. These compliment the new `dmgteam` monster key. Also made many changes to `monster` code.

Added `NO_STUPID_SPINNING`, `NO_DROPTOFLOOR`, and `SHOOTABLE` spawnflags to all `pickup items`.

Improved impact physics for `func_vehicle`. Also made it possible to damage `func_vehicle`.

Changed trigger code so that `sounds=3` (silent) works correctly. In normal Quake2, setting `sounds=3` sets the trigger to play the nonexistent `trigger1.wav`. It is silent, since the wav file doesn't exist, but an annoying error message is displayed at map startup.

June 4, 2000

I really don't care for idiot programmers who provide a constant barrage of "new and improved" updates, so I'll try not to become one. Having said that, a quick test by several folks at Rust turned up 3 fairly obnoxious problems that in my opinion were serious enough to justify an update. So here it is :-).

Bug Fixes

Fixed problem with `target_lock`. Maps may only have 1 `target_lock` with the HUD spawnflag set, for obvious reasons. However, in previous version if the player restarted a map after DYING (rather than using the map command), then the code erroneously counted the same `target_lock` again, reported an error, and destroyed the lock. This was a particularly stoopid error in the example map, since the HUD spawnflag isn't even supported in this version... what can I say? I was tired. I suppose this wasn't caught earlier because it was not possible to die in Tremor (where the `target_lock` originated).

Fixed a really nasty error with the `crane`, or actually with `func_pushables` being transported by the crane. In the previous (and this) version, stacked waterborne crates go through many gyrations in an attempt to unstack themselves. Depending on map geometry this may not be possible, but the stubborn crates will still try. In the previous version if you attempted to lift the top crate while this was going on, the crate would float up through the `crane_hook` and continue out of the level, and the crane was then completely flummoxed. In this version the crate touch function now checks to see whether a crate is being influenced by a crane, and bails out if so.

Changed the `func_vehicle` bounding box to rotate the extents as the vehicle rotates. In other words the bounding box will always be the smallest box that contains the vehicle. In previous version, the bounding box was initially squared off, making the small dimension equal to the large dimension. This mostly worked but was overly conservative (too large). Its harder now to get the vehicle stuck, but unfortunately still not impossible. Mapmakers should only use `func_vehicles` in fairly simple squared-off areas.

Other changes

Changed the method for taking control of and disengaging from `func_vehicle`. To drive the vehicle, hop on and press the

+use key once. To stop driving, press +use again. You will **not** be thrown from the vehicle as in the previous version.

Func_vehicle now damages monsters and opposing player (if moving). I'm sure we'll be playing around with the damage parameters a bit.

A couple of changes have been made to the example maps. In hyde2.bsp, I've added another crane_control that gives a better view of things (and illustrates that multiple crane_controls work :-)). In hyde3.bsp, the architecture has been changed a bit to make it harder to get the func_vehicle stuck.

June 1, 2000

Initial release

[Lazarus Main Page](#)

Lazarus Console

New console commands are available when running the **Lazarus** mod. Some are commands which are used for gameplay activity. Many others are developers' tools that were originally coded for no other reason than to help troubleshoot the new **Lazarus** entities during their development. We decided these last were useful enough to level makers to leave them in.

The following commands and variables are available *only* when the **Lazarus** gamex86.dll is loaded. This doesn't happen on Quake2 startup as you might expect; it happens when a map is loaded (at the point where you see "InitGame" at the console).

This is not a complete list of all Quake2 console commands and variables. For that you might want to check out JakFrost's herculean work at [The Console](#).

- Commands in **green** are *only* available when **developer** is **1**. (For more info on the use of **Lazarus** developer features, see [this page](#)).
- Commands in **blue** are really just aliased commands, and as such can be redefined by the user if so desired. Their initial definitions are set in **default.cfg** (found in Lazarus/PAK0.PAK).
- Commands in **yellow** are *only* available when **deathmatch** is **0**.

And now the list!

Lazarus Console Commands & Variables

+use/-use (action command)

When active, the player can activate certain triggers, buttons, etc; access remote turret controls; control the movement of pushable objects.

+zoom/-zoom (action command)

When active, sniper mode is on, with zoom-level fov equal to the current value of **zoomsnap**. Both mouse and game controller sensitivity scale with changes of zoom-level fov.

See also **+zoomin**, **+zoomout**, **zoom**, **zoomoff**, **zoomon**, **zoomsnap**, **zoomrate**.

+zoomin/-zoomin (action command)

When active, sniper mode is on, and field-of-view (fov) is decreased by **zoomrate** degrees/second, until the command is no longer active or until a limit of fov=5 is reached. When cancelled, the value of **zoomsnap** is updated. Exiting sniper mode is accomplished with **-zoom**.

See also **+zoom**, **+zoomout**, **zoom**, **zoomoff**, **zoomoff**, **zoomsnap**, **zoomrate**.

+zoomout/-zoomout (action command)

When active, sniper mode is on, and field-of-view (fov) is increased by **zoomrate** degrees/second, until the command is no longer active or until the user's default fov value is reached. When cancelled, the value of **zoomsnap** is updated, unless a +zoomout operation results in backing out to the user's default fov. Exiting sniper mode is accomplished with **-zoom**.

See also **+zoom**, **+zoomin**, **zoom**, **zoomoff**, **zoomoff**, **zoomsnap**, **zoomrate**.

actorchicken (toggle variable) (default=1)

Enables the use of AI "run and hide" code for **misc_actors** with low health. Changes to this variable do not take effect until the next map load.

0=disables actor AI-driven run & hide action.

1=enables actor AI-driven run & hide action.

See also **actorscram**.

actorjump (toggle variable) (default=1)

Enables the use of AI geometry checks so that **misc_actors** may jump over obstacles to get where they're going. This does not affect scripted jumping. Changes to this variable do not take effect until the next map load.

0=disables actor AI-driven jumping.

1=enables actor AI-driven jumping.

See also *monsterjump*.

actorscram (toggle variable) (default=1)

Enables the use of AI "evasive maneuvers" code for *misc_actors*. Changes to this variable do not take effect until the next map load.

0=disables actor AI-driven evasive action.

1=enables actor AI-driven evasive action.

See also *actorchicken*.

alert_sounds (toggle variable) (default=0)

Forces the worldspawn *ALERTSOUNDS effects* flag on. When this flag is set, monsters will be alerted to player footstep and falling sounds in addition to jump and weapon firing sounds.

allow_fog (toggle variable) (default=1)

Enables the use of *fog*. Has no effect in multiplayer, since fog is always disabled when deathmatch>0.

0=disables fog effects.

1=enables fog effects.

See also *fog*, *fog_blu*, *fog_density*, *fog_far*, *fog_grn*, *fog_help*, *fog_model*, *fog_near*, *fog_red*, *gl_driver_fog*.

bbox (command)

Draws blue-dotted lines outlining the bounding box of the entity you're currently looking at.

corpse_fade (toggle variable) (default=0)

Forces the worldspawn *CORPSEFADE effects* flag on. When set, monster, actor, and misc_insane corpses fade and sink into the floor after the time specified by *corpse_fadetime*.

corpse_fadetime (variable) (default=20)

Specifies the time, in seconds, before monster, actor, and misc_insane corpses fade away when the *CORPSEFADE effects* flag is set.

entlist (command) (syntax: *entlist [filename]*)

Outputs a listing of properties of all entities in the currently loaded map to the root QUAKE2 folder.

fmod_nomusic (variable) (default=0)

If non-zero, then target_playbacks with the *MUSIC* spawnflag will not be played. This allows the player the option of whether to listen to target_playback background music.

fog (toggle variable) (default=0)

Enables the rendering of fog.

0=fog off

1=fog on

See also *allow_fog*, *fog_blu*, *fog_density*, *fog_far*, *fog_grn*, *fog_help*, *fog_model*, *fog_near*, *fog_red*, *gl_driver_fog*.

fog_blu (variable) (default=0.5)

Sets the level of the blue color component of the active fog. Valid values are 0-1.

See also *allow_fog*, *fog*, *fog_density*, *fog_far*, *fog_grn*, *fog_help*, *fog_model*, *fog_near*, *fog_red*, *gl_driver_fog*.

fog_density (variable) (default=20)

Sets the overall level of fog density when *fog_model* is 1 or 2. For best results, *fog_density* should be less than 100.

See also *allow_fog*, *fog*, *fog_blu*, *fog_far*, *fog_grn*, *fog_help*, *fog_model*, *fog_near*, *fog_red*, *gl_driver_fog*.

fog_far (variable) (default=0)

Sets the distance from the player's viewpoint in map units that the fog will completely obscure the world when See also *allow_fog*, *fog*, *fog_blu*, *fog_density*, *fog_grn*, *fog_help*, *fog_model*, *fog_near*, *fog_red*, *gl_driver_fog*.

fog_grn (variable) (default=0.5)

Sets the level of the green color component of the active fog. Valid values are 0-1.

See also *allow_fog*, *fog*, *fog_blu*, *fog_density*, *fog_far*, *fog_help*, *fog_model*, *fog_near*, *fog_red*, *gl_driver_fog*.

fog_help (command)

Lists fog console commands. Fog commands allow the creation of fog effects in any map run in OpenGL or Glide, and are useful for tweaking values prior to compiling a map.

See also [allow_fog](#), [fog](#), [fog_blu](#), [fog_density](#), [fog_far](#), [fog_grn](#), [fog_model](#), [fog_near](#), [fog_red](#), [gl_driver_fog](#).

fog_model (variable) (default=1)

Sets how fog density will increase with distance.

0=linear

1=exponential

2=exponential squared

See also [allow_fog](#), [fog](#), [fog_blu](#), [fog_density](#), [fog_far](#), [fog_grn](#), [fog_help](#), [fog_near](#), [fog_red](#), [gl_driver_fog](#).

fog_near (variable) (default=0)

Sets the distance from the player's viewpoint in map units that the fog begins when [fog_model](#) is 0. Valid values are >0 and <[fog_far](#).

See also [allow_fog](#), [fog](#), [fog_blu](#), [fog_density](#), [fog_far](#), [fog_grn](#), [fog_help](#), [fog_model](#), [fog_red](#), [gl_driver_fog](#).

fog_red (variable) (default=0.5)

Sets the level of the red color component of the active fog. Valid values are 0-1.

See also [allow_fog](#), [fog](#), [fog_blu](#), [fog_density](#), [fog_far](#), [fog_grn](#), [fog_help](#), [fog_model](#), [fog_near](#), [gl_driver_fog](#).

footstep_sounds (variable) (default=0)

If non-zero, the worldspawn effects [STEPSOUNDS](#) flag is forced on, and the game will look for the file *texsurfs.txt* in the game folder. If present, footstep sound/texture name associations are read from the file. This cvar must be set before a map is loaded. For more information see the [footsteps](#) documentation.

freeze (toggle command)

Freezes all entities other than the player in place, including monsters, projectiles, gibbs, and brush models. Player cannot fire a weapon or touch a trigger while *freeze* is in effect. Note that effects like blood splatters and rocket trails are **not** affected in any way, since they are a fire-and-forget command sent to the executable. This command is useful for screenshots, and is the ancestor to a powerup that will be added to future versions.

go (command)

Used in conjunction with [GOOD_GUY](#) [misc_actors](#). To use, look at a *misc_actor* and type "go" at the console (or bind a key to *go*). The actor will run up to 256 units directly away from the player. This is useful for forcing a buddy actor to get out of the way, and with a bit of care can also be used to send an actor into a room ahead of you to take on any bad guys.

gl_driver_fog (variable) (default=opengl32)

Sets the name of the rendering .DLL that will be used to render fog effects. Ignored when "gl_driver" is "3dfxgl".

Useful when pointing the video subsystem to renamed .DLL's through console commands or config files rather than through the in-game video menu.

See also [allow_fog](#), [fog](#), [fog_blu](#), [fog_density](#), [fog_far](#), [fog_grn](#), [fog_help](#), [fog_model](#), [fog_near](#), [fog_red](#).

hint_test (toggle command)

Useful for testing the usage of [hint_paths](#) by monsters/actors. When the command is executed while looking at a particular monster/actor, that monster/actor will abandon normal AI routines and will become a hint_path-following automaton. Print messages will appear on the screen to let you know what the monster is "thinking" at any point. Execute the command a second time while looking at that same monster/actor will cause him to abandon his hint_path searches and revert to normal.

See also [medic_test](#).

hud (variable) (default=1)

Sets the on/off state of the HUD display. If **hud** is entered without a parameter, it will toggle the HUD. Useful for screenshots and demos. (A demo recorded with the HUD disabled will play back with the HUD disabled).

0=disables HUD display.

1=enables HUD display.

id (command)

Reports many properties of the entity you're currently looking at.

item_back (command)

Moves the entity the player is currently looking at one [shift_distance](#) unit in the direction away from him along either the X or Y axis, whichever is closer to being parallel to the player's current view direction.

See also [item_down](#), [item_drop](#), [item_forward](#), [item_left](#), [item_pitch](#), [item_release](#), [item_right](#), [item_roll](#), [item_up](#),

item_yaw, shift_distance.

item_down (command)

Moves the entity the player is currently looking at one *shift_distance* unit straight down (along the Z axis).

See also *item_back, item_drop, item_forward, item_left, item_pitch, item_release, item_right, item_roll, item_up, item_yaw, shift_distance.*

item_drop (command)

Drops the entity the player is currently looking at to the floor.

See also *item_back, item_down, item_forward, item_left, item_pitch, item_release, item_right, item_roll, item_up, item_yaw.*

item_forward (command)

Moves the entity the player is currently looking at one *shift_distance* unit in the direction toward him along either the X or Y axis, whichever is closer to being parallel to the player's current view direction.

See also *item_back, item_down, item_drop, item_left, item_pitch, item_release, item_right, item_roll, item_up, item_yaw, shift_distance.*

item_left (command)

Moves the entity the player is currently looking at one *shift_distance* unit in the direction to the left of him along either the X or Y axis, whichever is closer to being perpendicular to the player's current view direction.

See also *item_back, item_down, item_drop, item_forward, item_pitch, item_release, item_right, item_roll, item_up, item_yaw, shift_distance.*

item_pitch (command)

Rotates the entity the player is currently looking at one *rotate_distance* unit on the X axis, using the entity's own origin. If the entity is a brush model with no origin brush, then the brush model will be rotated around the map origin.

See also *item_back, item_down, item_drop, item_forward, item_left, item_release, item_right, item_roll, item_up, item_yaw, rotate_distance.*

item_release (command)

Releases control of the entity the player is currently looking at that has been manipulated with any of the other *item_** commands.

See also *item_back, item_down, item_drop, item_forward, item_left, item_pitch, item_right, item_roll, item_up, item_yaw.*

item_right (command)

Moves the entity the player is currently looking at one *shift_distance* unit in the direction to the right of him along either the X or Y axis, whichever is closer to being perpendicular to the player's current view direction.

See also *item_back, item_down, item_drop, item_forward, item_left, item_pitch, item_release, item_roll, item_up, item_yaw, shift_distance.*

item_roll (command)

Rotates the entity the player is currently looking at one *rotate_distance* unit on the Y axis, using the entity's own origin. If the entity is a brush model with no origin brush, then the brush model will be rotated around the map origin.

See also *item_back, item_down, item_drop, item_forward, item_left, item_pitch, item_release, item_right, item_up, item_yaw, rotate_distance.*

item_up (command)

Moves the entity the player is currently looking at one *shift_distance* unit straight up (along the Z axis).

See also *item_back, item_down, item_drop, item_forward, item_left, item_pitch, item_release, item_right, item_roll, item_yaw, shift_distance.*

item_yaw (command)

Rotates the entity the player is currently looking at one *rotate_distance* unit on the Z axis, using the entity's own origin. If the entity is a brush model with no origin brush, then the brush model will be rotated around the map origin.

See also *item_back, item_down, item_drop, item_forward, item_left, item_pitch, item_release, item_right, item_roll, item_up, rotate_distance.*

jetpack_weenie (variable) (default=0)

Toggles between normal, "manly" *jetpack* use and reducing it to a glorified "fly" cheat. In other words, you'll hover without applying thrust rather than falling. This is a server-side cvar.

0=disables jetpack cheat.
1=enables jetpack cheat.

jump_kick (toggle variable) (default=0)

Forces the worldspawn **JUMPKICK effects** flag on. When set, player can hurt damageable entities by jumping into them.

LAZBINDS (aliased command)

Loads the configuration file *suggested.cfg* if present. This is so **Lazarus**-specific gameplay key assignments can be made easily. This command is *not* case-sensitive.

LAZDEV (aliased command)

Alternately loads the configuration files *developer.cfg* (which loads developer binds & scripts) and *normal.cfg* (which, if properly configured, will undo what *developer.cfg* does), if present. This command is *not* case-sensitive. (Note: This command may not be present in all redistributions, since the level maker may well remove this command from *default.cfg* before distributing his level(s).

lightswitch (toggle command)

Mimics the effect of a **target_lightswitch**. When lights are turned off, the value of **lightsmin** is used.
See also lightsmin.

lightsmin (variable) (default=a)

Minimum light level switched to when the **lightswitch** command is used, or when a **target_lightswitch** entity is triggered. Valid values are a-z, where "a" is no light and "z" is overbright ("m" is normal light level).
See also lightswitch.

medic_test (toggle command)

Mimics the effect of triggering **monster_medic logic** in such a way that all currently spawned medics in a map will search for and follow **hint_paths** in order to find monster corpses to resurrect. Print messages will appear on the screen to alert you as to when the medic actually discovers corpses to revive, and if/when he decides to use **hint_paths** to get to them. Disabling **medic_test** will not cause such medics to abandon this behavior; once they're activated, that's it. The **medic_test** will be disabled automatically if another map is loaded.
See also hint_test.

monsterjump (toggle variable) (default=1)

Enables the use of AI geometry checks so that **monsters** may jump over obstacles to get where they're going. This does not affect scripted jumping. Changes to this variable do not take effect until the next map load.
0=disables monster AI-driven jumping.
1=enables monster AI-driven jumping.
See also actorjump.

muzzle (command)

Reports weapon-firing origin offset of the **misc_actor** you are currently looking at, if he is actively firing his weapon. If the actor has 2 weapon-firing origins, only the first one is reported.
See also muzzlex, muzzley, muzzlez.

muzzlex (variable)

Sets the weapon-firing origin offset on the X-axis of a **misc_actor**.
See also muzzle, muzzley, muzzlez.

muzzley (variable)

Sets the weapon-firing origin offset on the Y-axis of a **misc_actor**.
See also muzzle, muzzlex, muzzlez.

muzzlez (variable)

Sets the weapon-firing origin offset on the Z-axis of a **misc_actor**.
See also muzzle, muzzlex, muzzley.

packet_fmod_playback (variable) (default=0)

Toggles the use of **FMOD** sound file playing when using a **target_playback**. This is a server-side cvar. Ignored when **deathmatch=0**.
0 = disable FMOD playback

1 = enable FMOD playback
See also *playsound*.

playsound (command)

Plays a sound file one time. Similar to "play", but also supports playing files in the .MP3 and .MID format as well as the .WAV format. *Note: FMOD.DLL must be installed to the game directory in order for this level of sound playback support to function (see this page for details).*

Syntax: playsound [path/filename.extension]

See also *packet_fmod_playback*, *sound_restart*.

readout (toggle variable) (default=0)

Toggles displaying diagnostic information at 0.1 second intervals. Information displayed is dependent on the presence of certain active entities in the currently loaded map. At this time, enabling this cvar will display:

- The position and velocity of a misc_viper_bomb.
- The velocity and acceleration of an entity targeted by a *target_attractor*.
- Model and sound indices as they are used, as an Index:Overflow diagnostic (*Note: readout must be set to 1 prior to the loading of the Lazarus .dll for this particular feature to be used*).

0 = disable readout display

1 = enable readout display

rocket_strafe (toggle variable) (default=0)

Toggles the adding of player lateral and vertical velocity to player-fired rockets.

0 = disable adding player velocity

1 = enable adding player velocity

rotate_distance (variable) (default=1)

Distance in degrees that an entity will be rotated with any relevant item_* command operation. Fractional values are allowed. When the +use command is active, the item will be rotated continuously.

See also *item_back*, *item_down*, *item_drop*, *item_forward*, *item_left*, *item_pitch*, *item_release*, *item_right*, *item_roll*, *item_up*, *item_yaw*, *shift_distance*.

shift_distance (variable) (default=1)

Distance in map units that an entity will be moved with any relevant item_* command operation. Fractional values are allowed. When the +use command is active, the item will be moved continuously.

See also *item_back*, *item_down*, *item_drop*, *item_forward*, *item_left*, *item_pitch*, *item_release*, *item_right*, *item_roll*, *item_up*, *item_yaw*, *rotate_distance*.

sound_restart (command)

Restarts the sound subsystem. Used as an alternative to "snd_restart", which can create sound oddities when used on systems where both "s_primary" is set to 1, and the FMOD.dll is loaded either for maps that use a *target_playback* or when the *playsound* command is used.

See also *playsound*.

spawn classname (command)

Adds any point entity to a map, facing the same direction as the player and 64 units in front of the player. Look slightly up to prevent embedding spawned entities in the floor. The entity can then be moved around using the normal Lazarus *item movement commands*, and position info can be retrieved using the *id* command. Of course in the case of monsters you'll want to do this with *notarget* set.

spawnself (command)

Drops a copy of the player model into the map, facing the same direction as the player and 64 units in front of the player. If the player is currently using *notarget*, then monsters will pay no attention to the fake player. Otherwise monsters will consider the fake player fair game, just as they would a normal player. Fake player health is set to the player's health at the time the fake was spawned. This feature is primarily intended to assist with making action screenshots.

sv_maxgibs (variable) (default=20)

Sets the maximum number of gibbs that may spawn in any single game frame, in an effort to eliminate a possible cause of SZ_GetSpace: Overflow occurrences. This is a server-side cvar.

texture (command)

Reports texture name, surface flags, and value of the texture applied to the brush face you're currently looking at.

thirdperson (*toggle command*)

Toggles between first-person and third-person perspective.

See also *tp*, *tp_auto*.

tp (*persistant toggle variable*) (*default=0*)

Toggles between first-person and third-person perspective.

0 = first-person perspective

1 = third-person perspective

See also *thirdperson*, *tp_auto*.

tp_auto (*toggle variable*) (*default=1*)

Enables the automatic toggling between first-person and third-person perspective when a *func_pushable* is used.

0 = disables automatic toggling

1 = enables automatic toggling

See also *thirdperson*, *tp*.

zoom (*toggle variable*) (*default=0*)

Toggles sniper mode on/off (using the current *zoomsnap* value. Similar to *+zoom*, and is useful mainly for configuration scripts.

0 = sniper mode off

1 = sniper mode on

See also *+zoom*, *+zoomin*, *+zoomout*, *zoomoff*, *zoomon*, *zoomrate*, *zoomsnap*.

zoomoff (*command*)

Turns sniper mode off. Identical to *zoom 0*. Useful mainly for configuration scripts.

See also *+zoom*, *+zoomin*, *+zoomout*, *zoom*, *zoomon*, *zoomrate*, *zoomsnap*.

zoomon (*command*)

Turns sniper mode on. Identical to *zoom 1*. Useful mainly for configuration scripts.

See also *+zoom*, *+zoomin*, *+zoomout*, *zoom*, *zoomon*, *zoomrate*, *zoomsnap*.

zoomrate (*persistant variable*) (*default=80*)

Speed that *+zoomin* and *+zoomout* will change fov, in degrees/second.

See also *+zoom*, *+zoomin*, *+zoomout*, *zoom*, *zoomoff*, *zoomon*, *zoomrate*.

zoomsnap (*persistant variable*) (*default=20*)

Specifies the fov value to snap to when *+zoom* is active. This value is updated by *+zoomin* and *+zoomout* operations.

See also *+zoom*, *+zoomin*, *+zoomout*, *zoom*, *zoomoff*, *zoomon*, *zoomrate*.

DISCLAIMER

Neither David Hyde nor Tony Ferrara make any warranties regarding this product. Users assume responsibility for the results achieved for computer program use and should verify applicability and accuracy.

Contact



Have a comment, suggestion, gripe, rant, or the like? Feel free to address it to:

MrHyde - All coding issues. In other words, everything that's really important.
-=rascal@vicksburg.com==

Mad Dog - Website gripes, gameplay issues, etc. In other words everything that would otherwise distract MrHyde from making Quake2 jump through hoops.

Currently AWOL. Here's hoping everything works out for our pal and he's soon back here bugging the crap out of Mr. Hyde. In the meantime, please address all Lazarus mail to Mr. Hyde.

DISCLAIMER

Neither David Hyde nor Tony Ferrara make any warranties regarding this product. Users assume responsibility for the results achieved for computer program use and should verify applicability and accuracy.

[Lazarus Main Page](#)

Count key/value pair

Lazarus adds support for a **count** key to many entities. This can optionally enable the entity in question to use a sort of internal trigger_counter which will keep track of how many times the entity is used. The default value for **count** is zero; when count=0 it is as if no count is specified at all. When count>0, the value decrements by one for each use (see list below for details). When a count>0 value decrements to zero, the entity will be auto-killtargeted, which removes it from the map. There is a slight delay to the killtargeting effect, to allow the entity to perform its final action before it is removed. If the action takes some time to perform, like in the case of a **target_fog** or a **target_lightramp**, the killtargeting will not take place until the action is complete.

Removing entities that will never again be used from your map is a Good Thing™. This cuts down on processing overhead and reduces the number of active edicts. (Maximum allowable edicts at any one time in a Quake2 map is 1024). To do the same thing in normal Quake2 would require an additional trigger to cause the killtargeting event to happen. This would be pointless since net number of entities is therefore not reduced... worse, on map startup, this additional killtargeting entity exists, taking up an edict slot. Internalizing the killtargeting function by using **count** solves this.

How **count** works depends on the entity. The effect basically falls into three categories. For entities which fire and then sit idle until their next firing, **count** is decremented each time the entity's targetname is called. For entities which toggle on/off, **count** is decremented at each deactivation event. Other entities will see their **count** values decremented each time they are touched (the path_corner is an example of this). The list below details what entities currently enjoy support for this feature, and how it works for them.

The following entities will see their **count** values decremented to zero each time their targetnames are called. For example, a **target_help** with **count=1** will be auto-killtargeted after one use.

```
crane_reset
func_killbox
misc_viper_bomb (MULTI_USE only)
target_anger
target_animation
target_bmodel_spawner
target_cd
target_explosion
target_fade
target_fog
target_help
target_lightramp
target_lightswitch
target_monitor
target_monsterbattle
target_movewith
target_playback (doesn't apply to looping sounds)
target_rocks
target_rotation
target_sky
target_spawner
target_speaker (doesn't apply to looping sounds)
target_temp_entity
trigger_relay
```

The following entities will see their **count** values decremented to zero each time they are toggled **off**. For example, a **TOGGLE func_wall** with **count=1** will be auto-killtargeted after it is turned on, then off again. *Triggers with wait=-1 will be removed automatically, so there is no need to set a count for them.*

```
func_force_wall
```

- func_timer
- func_wall
- light
- target_attractor
- target_blaster (wait>0 only)
- target_laser
- trigger_bbox
- trigger_disguise
- trigger_fog
- trigger_look
- tremor_trigger_multiple

The following entities will see their **count** values decremented to zero each time they are touched. For example, a **path_corner** with **count=1** will be auto-killtargeted after it is touched once by the entity that is sent to it.

- info_player_start (only when deathmatch=0)
- path_corner
- point_combat (special case: see [this page](#))
- target_actor

[Lazarus Main Page](#)

Overhead (X-Y) crane



Lazarus' overhead crane consists of 4 separate entities plus 1 optional entity - crane_beam is the supporting mechanism that spans across a room. It typically moves along the length of a warehouse, supported by a beam/column system. crane_hoist does the lifting (in the real world), and moves back and forth along crane_beam. crane_hook is the lifting device; it moves vertically with horizontal movement relative to crane_hoist. crane_control provides the controls for the crane. It may be located in a separate area, or may move along with crane_beam. A crane can have multiple crane_controls. The optional crane_reset, if used, calls a crane to the crane_beam extent closest to the crane_reset. Before moving the crane, it checks to ensure that the crane_control is enabled.

For all but crane_control, each entity must have a pathtarget (NOTE: NOT target) path_corner. There should be 2 and only 2 path_corners in the sequence. These path_corners are only used to describe the direction of travel for the entity and the extents of the entity's movement... they are **not** used as waypoints. The crane orientation is limited to N-S or E-W, and the direction is determined by the relative magnitudes of the deltas between path_corners for crane_beam.

Target keys: Crane_control must target crane_beam, crane_beam targets crane_hoist, and crane_hoist targets crane_hook.

The entity will start in the same position it is drawn in the game. To move crane_control along with crane_beam, use the "team" key for these two entities. No other entity should use this team name. You can have multiple crane_controls for the same crane, but only one crane_control that moves along with the crane.

Sounds: Crane_hook plays a sound constantly if the associated crane_control is on (an electromagnet hum is nice). Use the noise field of the crane_hook to set the sound. The crane_beam noise moves along with crane_control (assuming it moves). The crane_hoist noise moves with crane_hoist.

Crane_control starts off unless spawnflags=1. If targeted, crane_control toggles on/off. If the player attempts to use crane_control while off, a message "No power" (or the message specified by crane_control's message key) is displayed.

Lights: Set the "spotlight" (=1) spawnflag for crane_hook to include a simulated spotlight that shines as the crane moves. The spotlight will help the player line the crane_hook up with the desired target. This isn't realistic, but the loss of perspective with a PC monitor (as opposed to the real world) makes this feature a practical necessity.

Crane_control controls all movement of the crane. Crane_control should use a single brush with one face that uses the crane/panel.wal texture or similar texture. The style key tells the game which direction is faced when looking out over the top of the control (0=east, 1=north, 2=west, 3=south)... and yeah we know there's an angle key, but it doesn't work properly in this case.... trust us on this. Controls are activated by looking at one of the psuedo-buttons and pressing the

+use key. These controls then target the appropriate crane entity, **if** the crane_control is 1) in front of the player, 2) visible to the player, 3) within 64 units of the player. Whether the panel.wal texture is used or not, the face of the control panel that faces the player is divided up (in the game's brain) into a matrix of 2 rows by 4 columns of psuedo buttons.

NOTE: Once a crane_control button is activated, the associated action continues as long as the player presses the +use key, **regardless of what the player is looking at**. This helps quite a bit when trying to move the crane hook to a desired location, when the hook would be out of the player's view if he were forced to look at the control panel.

When the pickup button is activated, the code checks underneath crane_hook for a **func_pushable**. If the distance to the func_pushable is less than X units (X is currently 64), the object "falls up" and will then move along with the crane until dropped.

Limitations:

The crane will only pick up one func_pushable at a time, and will not pick up a func_pushable with one or more other func_pushables stacked on top.

crane_control parameters

targetname

Name of the crane_control. Not optional if "Start On" spawnflag is not set. It is not strictly necessary, but to avoid confusion all crane_controls for a crane should have the same targetname.

target

Name of the crane_beam.

team

Name shared by crane_beam for onboard controls.

style

Direction faced by player when facing crane_control
0=East, 1=North, 2=West, 3=South

Spawnflags

Start On (=1). If set, crane is initially powered up. If not set, crane_control must be targeted. Crane power is toggled each time crane_control is triggered.

crane_beam parameters

targetname

Name of the crane_beam. This field should be used as the target of crane_control.

target

Name of the crane_hoist.

team

Name shared by crane_control for onboard controls.

speed

Speed of crane_beam in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving. Sound will emanate from a position near crane_control for onboard controls, and from end of crane beam closest to crane_control for crane_controls with fixed position.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hoist parameters

targetname

Name of the crane_hoist. This field should be used as the target of crane_beam.

target

Name of the crane_hook.

speed

Speed of crane_hoist in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hook parameters

targetname

Name of the crane_hook. This field should be used as the target of crane_hoist.

speed

Speed of crane_hook in units/sec

dmg

Damage to blocking entity

noise

Sound played when crane_control is powered up.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

Spawnflags

Spotlight (=1). If set, an invisible light-emanating sprite is projected below the crane_hook when any part of the crane is moving. This provides the player with visual cues missing because of the loss of perspective you get with a 2D monitor.

crane_reset parameters

targetname

Name of the crane_reset. Crane_reset must be targeted to function.

target

Name of crane_beam. When triggered, crane_reset calls the crane_beam to move to the extent closest to the crane_reset. In practice the crane_reset will normally be triggered by a func_button.

count

If non-zero, the value of **count** specifies the number of times the crane_reset can be used before it is auto-killtargeted.

Lazarus Main Page

Overhead (X-Y) crane



Lazarus' overhead crane consists of 4 separate entities plus 1 optional entity - `crane_beam` is the supporting mechanism that spans across a room. It typically moves along the length of a warehouse, supported by a beam/column system. `crane_hoist` does the lifting (in the real world), and moves back and forth along `crane_beam`. `crane_hook` is the lifting device; it moves vertically with horizontal movement relative to `crane_hoist`. `crane_control` provides the controls for the crane. It may be located in a separate area, or may move along with `crane_beam`. A crane can have multiple `crane_controls`. The optional `crane_reset`, if used, calls a crane to the `crane_beam` extent closest to the `crane_reset`. Before moving the crane, it checks to ensure that the `crane_control` is enabled.

For all but `crane_control`, each entity must have a `pathtarget` (NOTE: NOT `target`) `path_corner`. There should be 2 and only 2 `path_corners` in the sequence. These `path_corners` are only used to describe the direction of travel for the entity and the extents of the entity's movement... they are **not** used as waypoints. The crane orientation is limited to N-S or E-W, and the direction is determined by the relative magnitudes of the deltas between `path_corners` for `crane_beam`.

Target keys: `Crane_control` must target `crane_beam`, `crane_beam` targets `crane_hoist`, and `crane_hoist` targets `crane_hook`.

The entity will start in the same position it is drawn in the game. To move `crane_control` along with `crane_beam`, use the "team" key for these two entities. No other entity should use this team name. You can have multiple `crane_controls` for the same crane, but only one `crane_control` that moves along with the crane.

Sounds: `Crane_hook` plays a sound constantly if the associated `crane_control` is on (an electromagnet hum is nice). Use the noise field of the `crane_hook` to set the sound. The `crane_beam` noise moves along with `crane_control` (assuming it moves). The `crane_hoist` noise moves with `crane_hoist`.

`Crane_control` starts off unless `spawnflags=1`. If targeted, `crane_control` toggles on/off. If the player attempts to use `crane_control` while off, a message "No power" (or the message specified by `crane_control`'s message key) is displayed.

Lights: Set the "spotlight" (=1) spawnflag for `crane_hook` to include a simulated spotlight that shines as the crane moves. The spotlight will help the player line the `crane_hook` up with the desired target. This isn't realistic, but the loss of perspective with a PC monitor (as opposed to the real world) makes this feature a practical necessity.

`Crane_control` controls all movement of the crane. `Crane_control` should use a single brush with one face that uses the `crane/panel.wal` texture or similar texture. The style key tells the game which direction is faced when looking out over the top of the control (0=east, 1=north, 2=west, 3=south)... and yeah we know there's an angle key, but it doesn't work properly in this case.... trust us on this. Controls are activated by looking at one of the psuedo-buttons and pressing the

+use key. These controls then target the appropriate crane entity, **if** the crane_control is 1) in front of the player, 2) visible to the player, 3) within 64 units of the player. Whether the panel.wal texture is used or not, the face of the control panel that faces the player is divided up (in the game's brain) into a matrix of 2 rows by 4 columns of psuedo buttons.

NOTE: Once a crane_control button is activated, the associated action continues as long as the player presses the +use key, **regardless of what the player is looking at**. This helps quite a bit when trying to move the crane hook to a desired location, when the hook would be out of the player's view if he were forced to look at the control panel.

When the pickup button is activated, the code checks underneath crane_hook for a **func_pushable**. If the distance to the func_pushable is less than X units (X is currently 64), the object "falls up" and will then move along with the crane until dropped.

Limitations:

The crane will only pick up one func_pushable at a time, and will not pick up a func_pushable with one or more other func_pushables stacked on top.

crane_control parameters

targetname

Name of the crane_control. Not optional if "Start On" spawnflag is not set. It is not strictly necessary, but to avoid confusion all crane_controls for a crane should have the same targetname.

target

Name of the crane_beam.

team

Name shared by crane_beam for onboard controls.

style

Direction faced by player when facing crane_control
0=East, 1=North, 2=West, 3=South

Spawnflags

Start On (=1). If set, crane is initially powered up. If not set, crane_control must be targeted. Crane power is toggled each time crane_control is triggered.

crane_beam parameters

targetname

Name of the crane_beam. This field should be used as the target of crane_control.

target

Name of the crane_hoist.

team

Name shared by crane_control for onboard controls.

speed

Speed of crane_beam in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving. Sound will emanate from a position near crane_control for onboard controls, and from end of crane beam closest to crane_control for crane_controls with fixed position.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hoist parameters

targetname

Name of the crane_hoist. This field should be used as the target of crane_beam.

target

Name of the crane_hook.

speed

Speed of crane_hoist in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hook parameters

targetname

Name of the crane_hook. This field should be used as the target of crane_hoist.

speed

Speed of crane_hook in units/sec

dmg

Damage to blocking entity

noise

Sound played when crane_control is powered up.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

Spawnflags

Spotlight (=1). If set, an invisible light-emanating sprite is projected below the crane_hook when any part of the crane is moving. This provides the player with visual cues missing because of the loss of perspective you get with a 2D monitor.

crane_reset parameters

targetname

Name of the crane_reset. Crane_reset must be targeted to function.

target

Name of crane_beam. When triggered, crane_reset calls the crane_beam to move to the extent closest to the crane_reset. In practice the crane_reset will normally be triggered by a func_button.

count

If non-zero, the value of **count** specifies the number of times the crane_reset can be used before it is auto-killtargeted.

Lazarus Main Page

Overhead (X-Y) crane



Lazarus' overhead crane consists of 4 separate entities plus 1 optional entity - crane_beam is the supporting mechanism that spans across a room. It typically moves along the length of a warehouse, supported by a beam/column system. crane_hoist does the lifting (in the real world), and moves back and forth along crane_beam. crane_hook is the lifting device; it moves vertically with horizontal movement relative to crane_hoist. crane_control provides the controls for the crane. It may be located in a separate area, or may move along with crane_beam. A crane can have multiple crane_controls. The optional crane_reset, if used, calls a crane to the crane_beam extent closest to the crane_reset. Before moving the crane, it checks to ensure that the crane_control is enabled.

For all but crane_control, each entity must have a pathtarget (NOTE: NOT target) path_corner. There should be 2 and only 2 path_corners in the sequence. These path_corners are only used to describe the direction of travel for the entity and the extents of the entity's movement... they are **not** used as waypoints. The crane orientation is limited to N-S or E-W, and the direction is determined by the relative magnitudes of the deltas between path_corners for crane_beam.

Target keys: Crane_control must target crane_beam, crane_beam targets crane_hoist, and crane_hoist targets crane_hook.

The entity will start in the same position it is drawn in the game. To move crane_control along with crane_beam, use the "team" key for these two entities. No other entity should use this team name. You can have multiple crane_controls for the same crane, but only one crane_control that moves along with the crane.

Sounds: Crane_hook plays a sound constantly if the associated crane_control is on (an electromagnet hum is nice). Use the noise field of the crane_hook to set the sound. The crane_beam noise moves along with crane_control (assuming it moves). The crane_hoist noise moves with crane_hoist.

Crane_control starts off unless spawnflags=1. If targeted, crane_control toggles on/off. If the player attempts to use crane_control while off, a message "No power" (or the message specified by crane_control's message key) is displayed.

Lights: Set the "spotlight" (=1) spawnflag for crane_hook to include a simulated spotlight that shines as the crane moves. The spotlight will help the player line the crane_hook up with the desired target. This isn't realistic, but the loss of perspective with a PC monitor (as opposed to the real world) makes this feature a practical necessity.

Crane_control controls all movement of the crane. Crane_control should use a single brush with one face that uses the crane/panel.wal texture or similar texture. The style key tells the game which direction is faced when looking out over the top of the control (0=east, 1=north, 2=west, 3=south)... and yeah we know there's an angle key, but it doesn't work properly in this case.... trust us on this. Controls are activated by looking at one of the psuedo-buttons and pressing the

+use key. These controls then target the appropriate crane entity, **if** the crane_control is 1) in front of the player, 2) visible to the player, 3) within 64 units of the player. Whether the panel.wal texture is used or not, the face of the control panel that faces the player is divided up (in the game's brain) into a matrix of 2 rows by 4 columns of psuedo buttons.

NOTE: Once a crane_control button is activated, the associated action continues as long as the player presses the +use key, **regardless of what the player is looking at**. This helps quite a bit when trying to move the crane hook to a desired location, when the hook would be out of the player's view if he were forced to look at the control panel.

When the pickup button is activated, the code checks underneath crane_hook for a **func_pushable**. If the distance to the func_pushable is less than X units (X is currently 64), the object "falls up" and will then move along with the crane until dropped.

Limitations:

The crane will only pick up one func_pushable at a time, and will not pick up a func_pushable with one or more other func_pushables stacked on top.

crane_control parameters

targetname

Name of the crane_control. Not optional if "Start On" spawnflag is not set. It is not strictly necessary, but to avoid confusion all crane_controls for a crane should have the same targetname.

target

Name of the crane_beam.

team

Name shared by crane_beam for onboard controls.

style

Direction faced by player when facing crane_control
0=East, 1=North, 2=West, 3=South

Spawnflags

Start On (=1). If set, crane is initially powered up. If not set, crane_control must be targeted. Crane power is toggled each time crane_control is triggered.

crane_beam parameters

targetname

Name of the crane_beam. This field should be used as the target of crane_control.

target

Name of the crane_hoist.

team

Name shared by crane_control for onboard controls.

speed

Speed of crane_beam in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving. Sound will emanate from a position near crane_control for onboard controls, and from end of crane beam closest to crane_control for crane_controls with fixed position.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hoist parameters

targetname

Name of the crane_hoist. This field should be used as the target of crane_beam.

target

Name of the crane_hook.

speed

Speed of crane_hoist in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hook parameters

targetname

Name of the crane_hook. This field should be used as the target of crane_hoist.

speed

Speed of crane_hook in units/sec

dmg

Damage to blocking entity

noise

Sound played when crane_control is powered up.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

Spawnflags

Spotlight (=1). If set, an invisible light-emanating sprite is projected below the crane_hook when any part of the crane is moving. This provides the player with visual cues missing because of the loss of perspective you get with a 2D monitor.

crane_reset parameters

targetname

Name of the crane_reset. Crane_reset must be targeted to function.

target

Name of crane_beam. When triggered, crane_reset calls the crane_beam to move to the extent closest to the crane_reset. In practice the crane_reset will normally be triggered by a func_button.

count

If non-zero, the value of **count** specifies the number of times the crane_reset can be used before it is auto-killtargeted.

Lazarus Main Page

Overhead (X-Y) crane



Lazarus' overhead crane consists of 4 separate entities plus 1 optional entity - `crane_beam` is the supporting mechanism that spans across a room. It typically moves along the length of a warehouse, supported by a beam/column system. `crane_hoist` does the lifting (in the real world), and moves back and forth along `crane_beam`. `crane_hook` is the lifting device; it moves vertically with horizontal movement relative to `crane_hoist`. `crane_control` provides the controls for the crane. It may be located in a separate area, or may move along with `crane_beam`. A crane can have multiple `crane_controls`. The optional `crane_reset`, if used, calls a crane to the `crane_beam` extent closest to the `crane_reset`. Before moving the crane, it checks to ensure that the `crane_control` is enabled.

For all but `crane_control`, each entity must have a `pathtarget` (NOTE: NOT `target`) `path_corner`. There should be 2 and only 2 `path_corners` in the sequence. These `path_corners` are only used to describe the direction of travel for the entity and the extents of the entity's movement... they are **not** used as waypoints. The crane orientation is limited to N-S or E-W, and the direction is determined by the relative magnitudes of the deltas between `path_corners` for `crane_beam`.

Target keys: `Crane_control` must target `crane_beam`, `crane_beam` targets `crane_hoist`, and `crane_hoist` targets `crane_hook`.

The entity will start in the same position it is drawn in the game. To move `crane_control` along with `crane_beam`, use the "team" key for these two entities. No other entity should use this team name. You can have multiple `crane_controls` for the same crane, but only one `crane_control` that moves along with the crane.

Sounds: `Crane_hook` plays a sound constantly if the associated `crane_control` is on (an electromagnet hum is nice). Use the noise field of the `crane_hook` to set the sound. The `crane_beam` noise moves along with `crane_control` (assuming it moves). The `crane_hoist` noise moves with `crane_hoist`.

`Crane_control` starts off unless `spawnflags=1`. If targeted, `crane_control` toggles on/off. If the player attempts to use `crane_control` while off, a message "No power" (or the message specified by `crane_control`'s message key) is displayed.

Lights: Set the "spotlight" (=1) spawnflag for `crane_hook` to include a simulated spotlight that shines as the crane moves. The spotlight will help the player line the `crane_hook` up with the desired target. This isn't realistic, but the loss of perspective with a PC monitor (as opposed to the real world) makes this feature a practical necessity.

`Crane_control` controls all movement of the crane. `Crane_control` should use a single brush with one face that uses the `crane/panel.wal` texture or similar texture. The style key tells the game which direction is faced when looking out over the top of the control (0=east, 1=north, 2=west, 3=south)... and yeah we know there's an angle key, but it doesn't work properly in this case.... trust us on this. Controls are activated by looking at one of the psuedo-buttons and pressing the

+use key. These controls then target the appropriate crane entity, **if** the crane_control is 1) in front of the player, 2) visible to the player, 3) within 64 units of the player. Whether the panel.wal texture is used or not, the face of the control panel that faces the player is divided up (in the game's brain) into a matrix of 2 rows by 4 columns of psuedo buttons.

NOTE: Once a crane_control button is activated, the associated action continues as long as the player presses the +use key, **regardless of what the player is looking at**. This helps quite a bit when trying to move the crane hook to a desired location, when the hook would be out of the player's view if he were forced to look at the control panel.

When the pickup button is activated, the code checks underneath crane_hook for a **func_pushable**. If the distance to the func_pushable is less than X units (X is currently 64), the object "falls up" and will then move along with the crane until dropped.

Limitations:

The crane will only pick up one func_pushable at a time, and will not pick up a func_pushable with one or more other func_pushables stacked on top.

crane_control parameters

targetname

Name of the crane_control. Not optional if "Start On" spawnflag is not set. It is not strictly necessary, but to avoid confusion all crane_controls for a crane should have the same targetname.

target

Name of the crane_beam.

team

Name shared by crane_beam for onboard controls.

style

Direction faced by player when facing crane_control
0=East, 1=North, 2=West, 3=South

Spawnflags

Start On (=1). If set, crane is initially powered up. If not set, crane_control must be targeted. Crane power is toggled each time crane_control is triggered.

crane_beam parameters

targetname

Name of the crane_beam. This field should be used as the target of crane_control.

target

Name of the crane_hoist.

team

Name shared by crane_control for onboard controls.

speed

Speed of crane_beam in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving. Sound will emanate from a position near crane_control for onboard controls, and from end of crane beam closest to crane_control for crane_controls with fixed position.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hoist parameters

targetname

Name of the crane_hoist. This field should be used as the target of crane_beam.

target

Name of the crane_hook.

speed

Speed of crane_hoist in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hook parameters

targetname

Name of the crane_hook. This field should be used as the target of crane_hoist.

speed

Speed of crane_hook in units/sec

dmg

Damage to blocking entity

noise

Sound played when crane_control is powered up.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

Spawnflags

Spotlight (=1). If set, an invisible light-emanating sprite is projected below the crane_hook when any part of the crane is moving. This provides the player with visual cues missing because of the loss of perspective you get with a 2D monitor.

crane_reset parameters

targetname

Name of the crane_reset. Crane_reset must be targeted to function.

target

Name of crane_beam. When triggered, crane_reset calls the crane_beam to move to the extent closest to the crane_reset. In practice the crane_reset will normally be triggered by a func_button.

count

If non-zero, the value of **count** specifies the number of times the crane_reset can be used before it is auto-killtargeted.

Lazarus Main Page

Overhead (X-Y) crane



Lazarus' overhead crane consists of 4 separate entities plus 1 optional entity - `crane_beam` is the supporting mechanism that spans across a room. It typically moves along the length of a warehouse, supported by a beam/column system. `crane_hoist` does the lifting (in the real world), and moves back and forth along `crane_beam`. `crane_hook` is the lifting device; it moves vertically with horizontal movement relative to `crane_hoist`. `crane_control` provides the controls for the crane. It may be located in a separate area, or may move along with `crane_beam`. A crane can have multiple `crane_controls`. The optional `crane_reset`, if used, calls a crane to the `crane_beam` extent closest to the `crane_reset`. Before moving the crane, it checks to ensure that the `crane_control` is enabled.

For all but `crane_control`, each entity must have a `pathtarget` (NOTE: NOT `target`) `path_corner`. There should be 2 and only 2 `path_corners` in the sequence. These `path_corners` are only used to describe the direction of travel for the entity and the extents of the entity's movement... they are **not** used as waypoints. The crane orientation is limited to N-S or E-W, and the direction is determined by the relative magnitudes of the deltas between `path_corners` for `crane_beam`.

Target keys: `Crane_control` must target `crane_beam`, `crane_beam` targets `crane_hoist`, and `crane_hoist` targets `crane_hook`.

The entity will start in the same position it is drawn in the game. To move `crane_control` along with `crane_beam`, use the "team" key for these two entities. No other entity should use this team name. You can have multiple `crane_controls` for the same crane, but only one `crane_control` that moves along with the crane.

Sounds: `Crane_hook` plays a sound constantly if the associated `crane_control` is on (an electromagnet hum is nice). Use the noise field of the `crane_hook` to set the sound. The `crane_beam` noise moves along with `crane_control` (assuming it moves). The `crane_hoist` noise moves with `crane_hoist`.

`Crane_control` starts off unless `spawnflags=1`. If targeted, `crane_control` toggles on/off. If the player attempts to use `crane_control` while off, a message "No power" (or the message specified by `crane_control`'s message key) is displayed.

Lights: Set the "spotlight" (=1) `spawnflag` for `crane_hook` to include a simulated spotlight that shines as the crane moves. The spotlight will help the player line the `crane_hook` up with the desired target. This isn't realistic, but the loss of perspective with a PC monitor (as opposed to the real world) makes this feature a practical necessity.

`Crane_control` controls all movement of the crane. `Crane_control` should use a single brush with one face that uses the `crane/panel.wal` texture or similar texture. The `style` key tells the game which direction is faced when looking out over the top of the control (0=east, 1=north, 2=west, 3=south)... and yeah we know there's an `angle` key, but it doesn't work properly in this case.... trust us on this. Controls are activated by looking at one of the psuedo-buttons and pressing the

+use key. These controls then target the appropriate crane entity, **if** the crane_control is 1) in front of the player, 2) visible to the player, 3) within 64 units of the player. Whether the panel.wal texture is used or not, the face of the control panel that faces the player is divided up (in the game's brain) into a matrix of 2 rows by 4 columns of psuedo buttons.

NOTE: Once a crane_control button is activated, the associated action continues as long as the player presses the +use key, **regardless of what the player is looking at**. This helps quite a bit when trying to move the crane hook to a desired location, when the hook would be out of the player's view if he were forced to look at the control panel.

When the pickup button is activated, the code checks underneath crane_hook for a **func_pushable**. If the distance to the func_pushable is less than X units (X is currently 64), the object "falls up" and will then move along with the crane until dropped.

Limitations:

The crane will only pick up one func_pushable at a time, and will not pick up a func_pushable with one or more other func_pushables stacked on top.

crane_control parameters

targetname

Name of the crane_control. Not optional if "Start On" spawnflag is not set. It is not strictly necessary, but to avoid confusion all crane_controls for a crane should have the same targetname.

target

Name of the crane_beam.

team

Name shared by crane_beam for onboard controls.

style

Direction faced by player when facing crane_control
0=East, 1=North, 2=West, 3=South

Spawnflags

Start On (=1). If set, crane is initially powered up. If not set, crane_control must be targeted. Crane power is toggled each time crane_control is triggered.

crane_beam parameters

targetname

Name of the crane_beam. This field should be used as the target of crane_control.

target

Name of the crane_hoist.

team

Name shared by crane_control for onboard controls.

speed

Speed of crane_beam in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving. Sound will emanate from a position near crane_control for onboard controls, and from end of crane beam closest to crane_control for crane_controls with fixed position.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hoist parameters

targetname

Name of the crane_hoist. This field should be used as the target of crane_beam.

target

Name of the crane_hook.

speed

Speed of crane_hoist in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hook parameters

targetname

Name of the crane_hook. This field should be used as the target of crane_hoist.

speed

Speed of crane_hook in units/sec

dmg

Damage to blocking entity

noise

Sound played when crane_control is powered up.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

Spawnflags

Spotlight (=1). If set, an invisible light-emanating sprite is projected below the crane_hook when any part of the crane is moving. This provides the player with visual cues missing because of the loss of perspective you get with a 2D monitor.

crane_reset parameters

targetname

Name of the crane_reset. Crane_reset must be targeted to function.

target

Name of crane_beam. When triggered, crane_reset calls the crane_beam to move to the extent closest to the crane_reset. In practice the crane_reset will normally be triggered by a func_button.

count

If non-zero, the value of **count** specifies the number of times the crane_reset can be used before it is auto-killtargeted.

Lazarus Main Page

Overhead (X-Y) crane



Lazarus' overhead crane consists of 4 separate entities plus 1 optional entity - `crane_beam` is the supporting mechanism that spans across a room. It typically moves along the length of a warehouse, supported by a beam/column system. `crane_hoist` does the lifting (in the real world), and moves back and forth along `crane_beam`. `crane_hook` is the lifting device; it moves vertically with horizontal movement relative to `crane_hoist`. `crane_control` provides the controls for the crane. It may be located in a separate area, or may move along with `crane_beam`. A crane can have multiple `crane_controls`. The optional `crane_reset`, if used, calls a crane to the `crane_beam` extent closest to the `crane_reset`. Before moving the crane, it checks to ensure that the `crane_control` is enabled.

For all but `crane_control`, each entity must have a `pathtarget` (NOTE: NOT `target`) `path_corner`. There should be 2 and only 2 `path_corners` in the sequence. These `path_corners` are only used to describe the direction of travel for the entity and the extents of the entity's movement... they are **not** used as waypoints. The crane orientation is limited to N-S or E-W, and the direction is determined by the relative magnitudes of the deltas between `path_corners` for `crane_beam`.

Target keys: `Crane_control` must target `crane_beam`, `crane_beam` targets `crane_hoist`, and `crane_hoist` targets `crane_hook`.

The entity will start in the same position it is drawn in the game. To move `crane_control` along with `crane_beam`, use the "team" key for these two entities. No other entity should use this team name. You can have multiple `crane_controls` for the same crane, but only one `crane_control` that moves along with the crane.

Sounds: `Crane_hook` plays a sound constantly if the associated `crane_control` is on (an electromagnet hum is nice). Use the noise field of the `crane_hook` to set the sound. The `crane_beam` noise moves along with `crane_control` (assuming it moves). The `crane_hoist` noise moves with `crane_hoist`.

`Crane_control` starts off unless `spawnflags=1`. If targeted, `crane_control` toggles on/off. If the player attempts to use `crane_control` while off, a message "No power" (or the message specified by `crane_control`'s message key) is displayed.

Lights: Set the "spotlight" (=1) spawnflag for `crane_hook` to include a simulated spotlight that shines as the crane moves. The spotlight will help the player line the `crane_hook` up with the desired target. This isn't realistic, but the loss of perspective with a PC monitor (as opposed to the real world) makes this feature a practical necessity.

`Crane_control` controls all movement of the crane. `Crane_control` should use a single brush with one face that uses the `crane/panel.wal` texture or similar texture. The style key tells the game which direction is faced when looking out over the top of the control (0=east, 1=north, 2=west, 3=south)... and yeah we know there's an angle key, but it doesn't work properly in this case.... trust us on this. Controls are activated by looking at one of the psuedo-buttons and pressing the

+use key. These controls then target the appropriate crane entity, **if** the crane_control is 1) in front of the player, 2) visible to the player, 3) within 64 units of the player. Whether the panel.wal texture is used or not, the face of the control panel that faces the player is divided up (in the game's brain) into a matrix of 2 rows by 4 columns of psuedo buttons.

NOTE: Once a crane_control button is activated, the associated action continues as long as the player presses the +use key, **regardless of what the player is looking at**. This helps quite a bit when trying to move the crane hook to a desired location, when the hook would be out of the player's view if he were forced to look at the control panel.

When the pickup button is activated, the code checks underneath crane_hook for a **func_pushable**. If the distance to the func_pushable is less than X units (X is currently 64), the object "falls up" and will then move along with the crane until dropped.

Limitations:

The crane will only pick up one func_pushable at a time, and will not pick up a func_pushable with one or more other func_pushables stacked on top.

crane_control parameters

targetname

Name of the crane_control. Not optional if "Start On" spawnflag is not set. It is not strictly necessary, but to avoid confusion all crane_controls for a crane should have the same targetname.

target

Name of the crane_beam.

team

Name shared by crane_beam for onboard controls.

style

Direction faced by player when facing crane_control
0=East, 1=North, 2=West, 3=South

Spawnflags

Start On (=1). If set, crane is initially powered up. If not set, crane_control must be targeted. Crane power is toggled each time crane_control is triggered.

crane_beam parameters

targetname

Name of the crane_beam. This field should be used as the target of crane_control.

target

Name of the crane_hoist.

team

Name shared by crane_control for onboard controls.

speed

Speed of crane_beam in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving. Sound will emanate from a position near crane_control for onboard controls, and from end of crane beam closest to crane_control for crane_controls with fixed position.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hoist parameters

targetname

Name of the crane_hoist. This field should be used as the target of crane_beam.

target

Name of the crane_hook.

speed

Speed of crane_hoist in units/sec

dmg

Damage to blocking entity

noise

Sound played when moving.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

crane_hook parameters

targetname

Name of the crane_hook. This field should be used as the target of crane_hoist.

speed

Speed of crane_hook in units/sec

dmg

Damage to blocking entity

noise

Sound played when crane_control is powered up.

pathtarget

First in a sequence of 2 path_corners describing extents of travel.

Spawnflags

Spotlight (=1). If set, an invisible light-emanating sprite is projected below the crane_hook when any part of the crane is moving. This provides the player with visual cues missing because of the loss of perspective you get with a 2D monitor.

crane_reset parameters

targetname

Name of the crane_reset. Crane_reset must be targeted to function.

target

Name of crane_beam. When triggered, crane_reset calls the crane_beam to move to the extent closest to the crane_reset. In practice the crane_reset will normally be triggered by a func_button.

count

If non-zero, the value of **count** specifies the number of times the crane_reset can be used before it is auto-killtargeted.

Lazarus Main Page

Lazarus Credits



Of course none of this work would be possible without the efforts of id Software in developing Quake2 to start with, and more importantly allowing eggheads like ourselves access to the guts of the game.

Several features of Lazarus were taken and modified from features found in **MapPack...** as is the mod **philosophy** itself. Hats off to the MapPack crew for making source code available to the public.

Third-person perspective was developed by **Skull**.

Thanks to **Rroff** (Martin Painter) for providing his rotating `func_train` code, which provided a very good start for Lazarus' `func_train`.

Code for auto-generating modified id player models on-the-fly to support **misc_actors** using those models (and in such a way that id Software doesn't mind) **and** code for spline curves used by `func_train` and other movers was supplied by **Argh!** (Tim Wright).

Player models, skins, and sounds used in the **Lazarus ActorPak** are the property of their respective authors.

The jetpack code is modified from source developed by **Muce & Attila**, available from **QDeveLs**. The **jetpack** pickup model is from **ChaosDM**. The **ammo_fuel** model was created by **Privateer** (Paul Rogers). Jetpack sounds were created by **jeffrey** (Jeff Hayat) specifically for Lazarus.

Also from **jeffrey** are the excellent train sounds used by `func_tracktrain`.

Mud sounds were created by **Maulok** (Anthony Bouvette) specifically for Lazarus.

Venomus provided custom debris models for breakable stuff and a very cool pickup model and sounds for **item_freeze**. Thanks!

Func_reflect is based on original floor reflection code by **psychospaz**.

Security camera and gun turret models are from the excellent **Deadlode 2** mod. Other gun models and mounts from Deadlode 2 are included in the `lazarus2_XXXXXX` file on the **downloads** page.

QER entity file written by **Ricebug**, the only guy that can explain how to use QER in such a way that dopes like us can actually understand it.

The remaining code was written by Mr. Hyde. The majority of the strangely brilliant ideas implemented in Lazarus came from the strangely brilliant Mad Dog.

Many creative suggestions arose from the Tremor team:

- Bakersman (Matthew Baker)
- Divine Comedian (Ben Barker)
- gimp (Chris Brodie)
- Mad Dog (Tony Ferrara)
- Monsto Brukes (Darrin Lowery)

raYGunn (Jesse McCree)
Scampie (Sean Campbell)

We've received a lot of useful feedback and bug reports from beta testers **Argh!** (Tim Wright), **SoftShoulder** (Scott Kraus), **jeffrey** (Jeff Hayat), and **Jedi**. Thanks guys.

Special thanks to Monsto Brukes (Darrin Lowery) for being consistently obnoxious and prodding Mr. Hyde to do better work.

...and Lazarus wouldn't be what it is without the excellent suggestions we've received from idea man Mike Ferrara. Thanks Mike.

DISCLAIMER

Neither David Hyde nor Tony Ferrara make any warranties regarding this product. Users assume responsibility for the results achieved for computer program use and should verify applicability and accuracy.

[Lazarus Main Page](#)

Developer Tools

Lazarus offers a number of in-game diagnostic tools which can help you tweak entity settings and/or troubleshoot problems when building maps. Some of these features are useful even for maps that are intended to be run as standard **Quake2** levels. Many of these tools, which are **console commands** available only when running a **Lazarus** game, are enabled only when developer mode is turned on. If you've never used the developer cvar, either you've been very fortunate not to encounter any especially nasty bugs in Quake 2, or you just didn't know about it. Usually, this command is used to display diagnostic information that the normal player will never see.

```
developer 1 // turns developer mode on
developer 0 // turns developer mode off
```

Try it out some time :-).

The console buffer is limited, and besides, you might want to refer to the developer output at a later time. This can be done with the "logfile" cvar, which prints the output to a text file.

```
logfile 1 // writes a logfile
logfile 2 // appends an existing logfile
logfile 0 // don't write a logfile
```

Included with the main **Lazarus download** are a couple of configuration files, one of which is named `developer.cfg`. When loaded, it will not only turn on developer mode for you, but it will also assign many of the most-often used diagnostic stuff (already scripted) to function keys and keypad keys. The other configuration file, `normal.cfg`, is designed to turn developer mode off and rebind the same keys for gameplay use (but you'll have to edit this one, since we don't know what your gameplay binds are). These 2 files can be loaded alternately with the `LazDev` command (see the files themselves for details). This allows you to easily move from playtesting to diagnostic mode and back again.

More info on the current contents of `developer.cfg` appears [here](#).

Below are descriptions of the **Lazarus** developer features and suggestions for their use. All commands are entered at the console (or they may be bound to keys, or used in configuration scripts). For complete command & variable listings and definitions, here's yet another link to the [console doc](#).

- [Adjusting Light Levels](#)
- [Developer.cfg](#)
- [Entity Properties](#)
- [Fog](#)
- [Hint_path Testing](#)
- [In-Game Entity Manipulation](#)
- [Index:Overflow Diagnostics](#)
- [Medic Test \(hint_path usage\)](#)
- [Misc_actor Tweaking](#)
- [Texture Properties](#)
- [Velocity, Acceleration, and Position Data](#)

Adjusting Light Levels

The **lightswitch** command mimics a **target_lightswitch**. What it does is toggles between normal light levels and the light level specified by the **lightsmin** cvar, which uses a value range of a-z. Now, "lightsmin" is set to a by default, and normal light levels use a value of m. By changing the value of "lightsmin", you can use the **lightswitch** command to get an idea of what sort of light level

would look better than the one you're currently using in an area. For example, setting `lightsmin t` will let you toggle between normal light and a somewhat brighter light. This is certainly less time-consuming than doing a full radiosity compile just to see the effect of a slight change in the brightness of a few point lights.

Developer.cfg

Included with the main [Lazarus download](#) is a pair of configuration files, `developer.cfg` and `normal.cfg`. As mentioned above, the first file turns developer mode on and binds several keys with useful functions, and the second turns developer mode off again and unbinds those keys. The 2 files can be loaded alternately with the `LazDev` command, which can be bound to a key to facilitate this. Uncommenting a single line in `suggested.cfg` (also included in the download archive) will bind this command to the **F8** key.

Currently, this is what `developer.cfg` does, and what keys the commands are mapped to:

F5	Draws the bounding box of the currently viewed entity (bbox).
F6	Show entity information. While the key is held, the info will stay on the screen for a maximum of 60 seconds (id).
F7	Show texture information. While the key is held, the info will stay on the screen for a maximum of 60 seconds (texture).
F9	Test <code>hint_path</code> usage. Press this key while looking at a monster will cause him to search for and follow <code>hint_paths</code> . Press the key again while looking at the monster will restore his normal AI. (hint_test).
F10	Test medic logic and his <code>hint_path</code> usage. Press this key and all currently spawned medics in a map will revive any dead monsters they see, and/or will follow <code>hint_paths</code> to find more dead guys. Pressing the key again will deactivate this effect, but will not cause currently spawned medics to stop this behavior. (medic_test).
KeyPad Arrowkeys	Moves entity forward/back and left/right.
KeyPad Home KeyPad End	Moves entity up and down.
KeyPad PgUp KeyPad-5 KeyPad PgDn	Rotates entity along pitch, yaw, and roll axes, respectively.
KeyPad Enter	Reverses rotational direction for the rotation keys.
KeyPad Del	Drops entity to the floor (won't work for pickups with <code>NO_DROPTOFLOOR</code> spawnflag set).

Entity Properties

Many keyvalues of the entity you're looking at can be referenced while in the game by using the **id** command, which lets you check if targets are properly set, etc. Further, you can see a

representation of the bounding box of the entity you're looking at with the **bbox** command, which can be more useful than you might think. Try them out; you may consider these no more than a novelty at first, but as time goes on you'll come to appreciate them.

Fog

There's no need to do trial-and-error compiling to discover your "best" fog settings when using the **target_fog** and **trigger_fog** entities. Fog can be activated simply by entering **fog 1** ("fog 0" turns it off). You can then adjust the color components and the fog's density until you get it just the way you like it. Once you do, just assign those values to the entity in your editor, and no surprises. Also, **fog_help** will show you the available commands and their usage, in-game.

- Show me the fog console commands -

Hint_path Testing

So you've carefully laid **hint_path** chains in your map, but how do you know the monsters will actually use them? The **hint_test** command will tell you this, and will tell you where potential problems lie. Execute the command while looking at a monster/actor, and his normal AI is disabled, transforming him into a single-minded hint_path-following automaton. Print messages will appear on the screen to let you know what hint_path he's looking for, and when he's reached them. Execute the command a second time while looking at the affected monster/actor, and he'll stop his searches and revert to normal AI. Developer mode needs to be on to use this, and for best results, "notarget" should be on as well.

In-Game Entity Manipulation

Visible and solid entities can be picked up, moved around, and rotated on all 3 axes while in the game. If you've tried to make full use of the **NO_DROPTOFLOOR** and **NO_STUPID_SPINNING** spawnflags for realistically positioning **items**, **ammo**, **weapons**, and **keys**, you know what a trial-and-error pain in the butt that can be. By availing yourself of the **host of item-moving commands**, you can do the same by placing the item(s) in your map with the proper spawnflags set (just put 'em in the general vicinity of where you want them to go). Then do a bare-minimum compile (QBSP3 only will do), load the map, and move & orient the items to your liking. When satisfied, use the **id** command to get a reading of the entity's origin and angles value. Then, in your editor, position the entity accordingly, give it the proper angles value, and if done properly you'll find everything placed just as you wanted it the next time you compile your map.

There are more uses for this tool. You can reposition monsters (you must have **notarget** on of course), write a savegame, and then see how things play out with them in their new locations. If things are good, reload the savegame and take some **id** readings. You can move brush entities too. This makes it very easy to check on texture alignment in tight places, like lifts in elevator shafts.

Closely related are the **spawn** and **spawnself** developer commands. The "spawn" command can be used to drop any point entity in a map, directly in front of the player and facing the same direction faced by the player. "Spawnself" drops a copy of the player model into the map, which might be useful for screenshots. Of course any entity spawned into the game in this way may be manipulated with the item movement commands.

Some tips:

- Moving pickups is best done with **noclip** on, so you don't inadvertently pick up the item you're moving.

- When a point entity's origin is moved so that it's buried in a brush, you can't "see" it anymore, as far as the item-moving commands go. You *may* be able to "see" it again if you noclip into the brush it's buried in and look in its direction (it won't be drawn while you're outside of the level, so this isn't the easiest thing to do). Sometimes you just have to reload the map and try again.
- It can be difficult to move only the point entity you intend to move, if it's very close to other point entities - you might move them all. Try and position yourself (noclip) so as to only be looking at the entity you want to move.
- All entities rotate around their origin. A brush model without an origin brush will therefore rotate around the map origin (0,0,0). Try it on a bmodel far from the map origin; you'll see what happens.
- Entities that are moving between path_corners may be difficult to move, since they have their own ideas of where they want to be.
- Just because an entity is not drawn does not mean you can't move it. A nodraw, yet **solid** bmodel is moveable because it's solid. An example of this would be a func_wall with the clip texture, if it also had the window content property so as to block shots.

All item-moving commands are included in scripted form and mapped to the keypad in `developer.cfg`, in an attempt to make their use what we hope is at least somewhat intuitive.

- Show me the item-moving console commands -

Index:Overflow Diagnostics

If `readout=1`, **Lazarus** will display model and sound indices as they are used (image indices are not displayed since as far as we've seen, it's pretty tough to have too many images). Readout **must** be set on the command line or in a .cfg file which is loaded prior to the loading of the **Lazarus** .dll; once the game starts it is too late. For best results start the game with "+set logfile 2 +set readout 1". After walking through the map, go to a DOS prompt in the game folder and type "sort qconsole.log >index.txt" (or whatever name you like) to make the list a bit easier to read.

Medic Test (hint_path usage)

The `medic_test` command toggles automatic hint_path usage by the monster_medic, so you can see if he'll actually run around and heal guys as he's *intended* (your map will have to have `hint_paths` and medics, though, or nothing will happen). Here's how to use it: load your map with "notarget" on. Go around and kill everyone (but don't gib them), except the medics of course. Set developer to 1, and enter "medic_test". You should get a console message that medic_test is on. Any medics in the map will act as if they've been triggered but can't see the player, so they'll look for hint_paths. Follow them around and see if they're doing their jobs as you envisioned. Print messages will appear on the screen to let you know exactly when the medic sees a corpse to revive, and if/when he decides to use hint_paths to reach him.

Misc_actor Tweaking

The `misc_actor` entity allows the choice of any player model. But no two player models are the same. The most obvious differences are their sizes and animations. The **Lazarus** `misc_actor` defaults to using variously-sized bboxes for the `ActorPak` player models, and defaults to the standard player bounding box for "unknown" player models. For one reason or another, you may wish to use another bounding box size. To test your work, you can use a `model_spawn` to display all of the actor model's animations, and examine them while using the in-game `bbox` command.

If using an "unknown" player model (not in the `ActorPak`), then the actor's weapon firing origin is

almost certainly another area that will need adjustment. To discover what **muzzle** and **muzzle2** values to set, **Lazarus** offers the ability to easily see and adjust weapon firing origins in-game. With developer mode on, **misc_actors** will display a blue debugtrail which is drawn from the muzzle origin to the actor's target.

This is probably the best way to set up a muzzle-tweaking session: Make a test map with the actor you wish to tweak set to attack another actor. This is best done with a HOLD **target_actor** or a HOLD **target_anger**. The actor to be attacked should have his health set to 100000 so he'll never die. If you do this properly, the actor you're tweaking will stand in place and attack the second actor constantly.

While looking at the shooting actor, enter **muzzle**, and you'll get a reading of what the current firing origin offset is. These values can be changed in-game by assigning new values to the **muzzlex**, **muzzley**, and **muzzlez** cvars. When you're satisfied with the muzzle position, enter **muzzle** again and jot down the values shown for later use in your editor.

Only one weapon firing origin at a time can be tested this way, so test 2-gun actor models as single gun actors, and use the above procedure to discover the values you need for the **muzzle** entity property, and repeat for **muzzle2** entity property. Now you'll have the 2 sets of values you need to set up your 2-gun actor properly.

- Show me the muzzle console commands -

Texture Properties

You can discover the texture name, surface flags, and value number of brush faces in the game by using the **texture** command. Execute this command while looking at the selected brush face, and this data will appear on-screen. Much better than laboriously checking face after face with an editor's surface inspector, or sifting through hundreds of textures with a texture browser trying to discover the particular one that you saw used in a level.

Velocity, Acceleration, and Position Data

At certain points in **Lazarus's** development, it was desirable to see exactly how an entity was acting in-game. So, certain specific data readings were added so as to better observe these entities in action. Rather than remove this code, it's been retained since it can still be useful for the level maker when setting the values of these entities. Since the readings are reported at 0.1 second intervals, they unfortunately tend to spam the console big time, which causes all other developer messages to be lost in the shuffle. This is why these readings do not appear simply by enabling developer mode; it's necessary to enable the **readout** cvar as well. To turn it on, enter **readout 1**, and as you would expect, **readout 0** turns it off. In practice, making use of this diagnostic information is best done in small test maps, rather than in nearly-completed maps where a lot may be going on at once.

Currently, enabling **readout** will:

- Let you see the velocity and position of a **misc_viper_bomb** (helps you hit that target).
- Report the velocity and acceleration rate of the target of a **target_attractor** (helps set the most effective speed and accel values).
- Report items in the model and sound indices as they are used (helps diagnose Error: Index Overflow problems). *Readout must be set to 1 prior to loading the Lazarus game .dll (prior to "InitGame") in order to use this function.*

Lazarus Main Page

Dmgteam key/value pair

How many times have you picked off a distant enemy with the railgun, while his buddies standing next to him fail to react to the fact that one of their number has been reduced to a pile of gibs? How many times have you shot and injured a guy, and watch him throw a fit trying to get to you while his pals stand around oblivious? This is not a failing limited to Quake2 - first-person-shooter games have been (and still are) plagued with this idiocy.

That's where the **Lazarus** "dmgteam" key comes in. Whenever a **monster** or a **misc_actor** is hurt, he essentially fires an internal **dmgteam** trigger, if set. This trigger will activate all other monsters/actors which share the same dmgteam value. If the player shoots one dmgteam-mate, all of his buddies on the same dmgteam will act as if you shot them too.

This means you can construct rooms where AMBUSH/SIGHT monsters are placed realistically, rather than have everyone face the door waiting for the player. They can hide in alcoves, have their backs to the player, and can even be on the opposite side of closed doors. In normal Q2, a well-armed player who comes upon such a room would be able to pick off the occupants at his leisure. No more - put all these guys on the same dmgteam, and when the player shoots one monster, the rest will go apeshit. Guys in the next room, if placed properly, will open the door and burst in. This is such a refreshing difference from the way monsters act in the regular game, that once you start using dmgteam, you'll never go back.

Dmgteam can be used to trigger events as well. In the past, only monster deaths could be used to trigger anything (deathtarget); now the trigger action can take place at first blood. This can be done by using a **trigger_relay** - since trigger_relays enjoy the use of the dmgteam key as well. If the relay shares the same dmgteam value as a monster that the player injures, that relay will fire its targets. (It's probably a good idea to set such a trigger_relay's **count** to 1, or else the relay will fire with every shot each monster on the dmgteam takes).

A mixed group of monster classes can all share the same dmgteam and they'll act as you'd expect. Different monster classes will still fight among each other if damaged by friendly fire, even if they're on the same dmgteam. If a monster on a dmgteam is injured by friendly fire originating from a monster not on that dmgteam, the dmgteam-mates of the first monster will not break off their attack on the player to come to their dmgteam-mate's aid. This is so monster groups don't inexplicably (from the player's point of view) degenerate into having gang wars in the middle of battle with the player.

Dmgteam is a simple concept (but not-so-simple coding), with many applications, and is a very powerful tool in making the monsters/actors in your map appear to be a lot smarter and alert, while improving gameplay as well.

Lazarus Downloads



For the latest available files, please visit the [Lazarus Downloads](#) page at the website. You can sign up to be alerted to interim beta releases there as well.

[Lazarus Main Page](#)

Fog effects

Lazarus supports fog effects in any map which is run using the **Lazarus** gamex86.dll. The effect may be activated at will with a **console command**, and its density and RGB color components may be manipulated as well in the same manner. For maps made to take advantage of the **Lazarus**, the **target_fog** and **trigger_fog** entities are available, so as to introduce and/or change fog effects during gameplay. The above-mentioned console commands allow for in-game tweaking of fog settings without having to resort to trial-and-error map recompiling; see **these tips**.

When setting fog rendering properties, you can choose from among 3 different fog density models, which determine how fog density increases with view distance. There is a linear model, and two exponential models. Fog may also be given a "directional" effect, in that when standing in a given location and looking in one direction, the player will see one level of fog density, and will see a different level of density when looking in the opposite direction.

Fog density and color can be made to change as the player plays through the map. Such changes can be made to gradually ramp up and down over time rather than display abrupt unnatural changes; the amount of time required for such a ramp is another property of **Lazarus** fog entities which is available to you.

There are a couple of concepts that should be made clear:

- Fog doesn't "turn off" - instead it **changes**. It requires one fog entity to direct the game to render fog, and another fog entity to direct the game to stop rendering fog (in other words, render zero fog).
- Fog works by causing the effect to be displayed from the player's point of view, rather than creating any sort of fog "zones" which can be viewed from the outside. This is a limitation of the game engine. Since it is a client-side rendering effect, it can be easily disabled by the user, which makes it unsuitable for fair play in multiplayer games.

Given the above, you can see that showing the player believable fog requires that you either give a map a global fog effect and never change it, or change the fog effect as the player moves through the map by using carefully-placed triggers, being mindful of the areas the player can access versus what the player can see. It makes no sense for the player to see an area far away which is clear of fog, and then let him find dense fog when he gets there.

So, if you incorporate fog changes in your map, you should be employing whatever tricks you can think of to control the player's movement and view when the fog actually changes. One method would be to change the fog the player sees while he's riding a lift in an elevator shaft - such a situation would be ideal, since the player's movement is controlled over the course of the lift ride, and the transit time is known. This is of course only one example, and other techniques when applied to specific cases would no doubt work well also. Whatever mechanism you go with, be sure and playtest the result from every angle to ensure that the player can't do something which would end up leaving him with a fog effect that is not what you intend.

Hardware Considerations

Fog effects require 3D video acceleration. No problems will be encountered running maps with active fog in software mode - you just won't see any fog. **Lazarus** fog works under both the OpenGL and Glide API's - so that 3dfx users can run the game using the 3dfx miniGL and see the fog effects just fine. If your video card limits you to rendering the game in OpenGL, you should see nice fog as well. However, we have noted in the past that there can be some differences in how fog effects are rendered using different video hardware, and steps have been taken to improve this. But, since effects rendering is very dependent on the video drivers used, new driver revisions may result in some quirks if not just break things outright. If problems are encountered, then please gather all pertinent information together and **let us know**.

Fog does add computational overhead. In an active map, and/or a map with high r_speeds, you may find this overhead to be unacceptable. Keep in mind what the "target machine" is for your map, and add fog effects with discretion. Generally, limiting fog rendering to small areas causes no appreciable slowdowns. Fog will not have any impact on software-only systems, since they can't render fog to begin with.

3dfx Voodoo2/3/4/5 users please note: There's a possible software crash for V2 users and a hardware crash for the rest waiting to happen if you switch from "Default OpenGL" (gl_driver=opengl32) to "3dfx" (gl_driver=3dfxgl) while fog is

on. The reverse isn't true... switching to "Default OpenGL" causes no problems. What happens when changing video in Q2 is that the "gl_driver" setting is updated before the actual "vid_restart" event, so for 3dfx users, the code will try to send Glide instructions to the video card before it is ready to render something like fog. To avoid this, **Lazarus** attempts to preserve the frame number when using software rendering or when OpenGL or Glide fog is used, and will NOT use Glide fog calls until at least 1 second has passed since the last time the code detected software or OpenGL. When switching to the 3dfx minidriver you may notice a short hiccup with no fog, followed by the correct fog. As with all hardware-dependent things, your mileage may vary. If you experience a crash when switching "gl_driver" when a map with visible fog is running (like when checking your map by using **gl_showtris**), then please **alert us** to this fact, along with all pertinent video hardware/driver info.

Working with Lazarus Fog

As with everything else, there are limitations. Rather than let you beat your head against your monitor trying to get something to work, we figured we'd just clue you in on what some of the more relevant limitations are:

- Fog **console commands** are unavailable unless developer=1.
 - Total allowable fog entities in a given map is 15. A 16th fog entity is reserved to handle the fog **console commands**.
 - If the player is inside of a **trigger_fog** field, the fog specified by that trigger_fog will take precedence over any fog effect specified by a **target_fog**. When the player leaves the trigger_fog field, whatever fog settings (if any) will be again used.
 - In a similar manner, if the player is inside of a **trigger_fog** field, fog **console commands** will not work very well, if at all.
 - Ramping fog effects from one setting to another works a lot better when fog models are not mixed. Linear and exponential models use different parameters, and when one model type is in effect, the parameters of the other type are undefined. So the code has no clue of what values to ramp **from** when going from one model type to another.
-

Using Fog to Hide Visual Problems/Improve Performance

If you own a 3D card (and if you've read this far, chances are you do), then you've likely run into distance clipping problems in Quake2 before, either in your own maps or in maps created by others. It's a bit ironic that distance clipping does **not** take place using software rendering, but is a problem using 3D cards that are easily capable of displaying 10 times as many or more polygons than the software renderer can. What if you could create arbitrarily large outdoor maps (still restrained to ± 4096 limits of course) with a clear line of sight from one side of the map to the other and not worry about distance clipping? Would that make you happy? Yeah we thought so :-). To eliminate the nasty poly-dropping HOM-producing distance clipping problem from your map, use a target_fog that completely obscures visibility at a distance of roughly 2048 units, and voila... no more clipping problems.

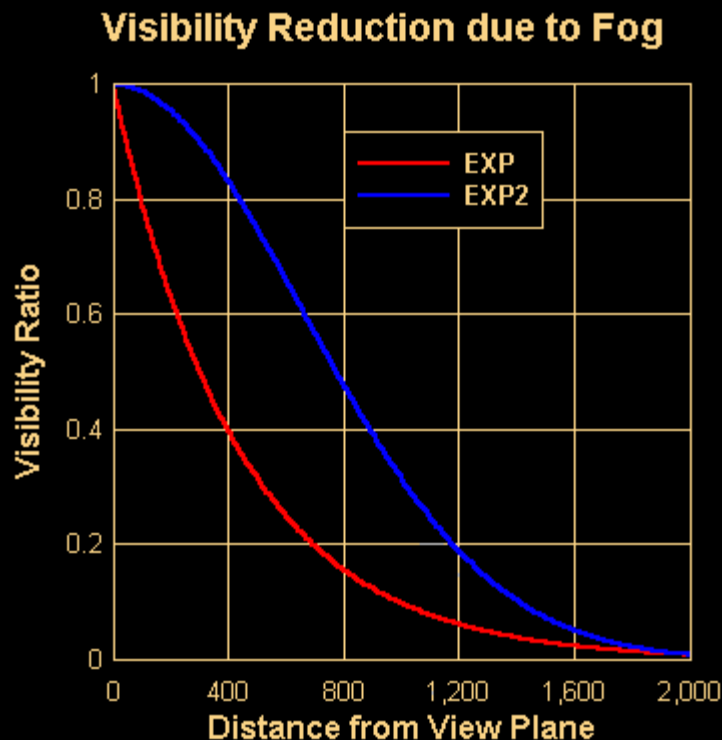
The **LINEAR fog model** is the obvious choice here, but you may not be happy with the appearance of that fog. As the name implies, visibility is reduced in a linear fashion from the fog_near to the fog_far distances, and at near distances the fog may be more dense than you'd like. So... use a larger fog_near value, they say. Well that kinda works, kinda doesn't. Large fog_near values tend to look very strange as the player turns his head from side to side... try it and you'll see. So let's take a look at the EXP and EXP2 models and see how they work. As the names imply, visibility with these models drops off exponentially with distance. Visibility is proportional to $\exp(-dw)$ for EXP, and $\exp((-dw)*(dw))$ for EXP2, where d is density (Lazarus fog_density/10000) and w is distance from the viewing plane. For all of you gearheads out there, yes it's obvious that the exponential models will never yield 0 visibility. However, for our purposes it's safe to reduce visibility to roughly 1%, and the fog will properly hide the visual problems introduced by distance clipping. So what fog_density values do we need to use to produce a given visibility level at a specified distance? You could try trial-and-error, of course, but that's not really necessary. Inverting the equations above, we get $d = -\ln(v)/w$ for the EXP model, and $d = \sqrt{-\ln(v)}/w$ for the EXP2 model, where v is the desired visibility ratio.

Asleep yet? OK wake up, here's the bottom line for the **fog_model/fog_density** values you're looking for to achieve 1% (0.01) visibility at a range of 2000 units:

- When **fog_model=1**: set **fog_density** to **23**

- When **fog_model=2**: set **fog_density** to 11

Visibility curves for these density settings are shown below. You'll notice that the EXP2 model yields greater visibility at close ranges, which... depending on what effect you are trying to achieve, may be what you want.



One remaining issue is the user's *gl_clear* setting. Unfortunately there is an inconsistency between OpenGL and 3dfx cards here: For OpenGL cards, *gl_clear* should be set to 1; otherwise users will still see HOM at the clipping distance. However, using *gl_clear*=1 for 3dfx cards results in a nice hot pink, while setting it to 0 yields the correct fog-shaded background. We've added a **fogclip** property to worldspawn, to assist you with automatically setting *gl_clear* to the correct value, on a map-by-map basis. If *gl_driver*="3dfxgl", then "fogclip 1" forces *gl_clear* to 0; otherwise it forces *gl_clear* to 1. If fogclip is 0 or not specified in the next map loaded, the user's previous *gl_clear* setting is restored.

But **wait!**... we're not done! This is where things start to get interesting :-). If you have Geoffrey DeWan's newest version of **qvis3**, you can provide a *maxdist* parameter which tells qvis3 that visibility nodes farther apart than this distance cannot see each other, whether they really can or not. What's this good for? Well you guys that create huge outdoor levels already know the answer, but for you normal people, the answer is that you'll be able to drastically reduce r_speeds in large open areas, using fog to hide the HOM resulting from this reduction. You'll of course be limited to producing maps for 3D card users only, and you'll need to either use a permanent fog setting with a **START_ON** target_fog, or if you want to turn fog off it may only be done in areas where the line-of-sight is less than *maxdist*. Turok for Quake2, anyone?

Footsteps

Lazarus allows you to play surface-specific footstep sounds for the player. New footstep sounds may be included in either or both of two ways:

1. Set the worldspawn **effects** STEPSOUNDS flag. Brush faces that use one of the new surface flags described below will cause the appropriate footstep sounds to play. In addition, sloshing through shallow water, wading through deep water, and ladder climbing sounds will be played at the appropriate times, regardless of surface flags.
2. Set the **footstep_sounds** cvar to 1 on the command line, and include a table of footstep sound indices for texture names in the file *texsurfs.txt* in the game folder. More on this below. Again, sloshing, wading, and ladder sounds will be played regardless of entries in *texsurfs.txt*. If **footstep_sounds** is set, the STEPSOUNDS flag is automatically turned on, and footstep surface flags will override any entries in *texsurfs.txt*. There are pros and cons to using this method over STEPSOUNDS. On the plus side, it can be used in **any** map, including those from the original game. WorldCraft users will like this method because it does not require editing the map file to include new surface flags. On the minus side, it's a bit more trouble to carefully control which surfaces will play which sounds. For example, you might specify that all textures containing the character string "metal" play metal sounds. But in many cases the metal textures are used for surfaces that don't appear to be metal at all. The other drawback to this method is that you cannot distinguish between specific surfaces that use the same texture - all surfaces that use a texture included in *texsurfs.txt* will play the corresponding footstep sound.

One of these two methods is required for a couple of reasons:

1. The wading sounds (sound/player/wadeX.wav) use the standard Q2 sound system, so we're adding 3 new sounds to the total number of unique sounds. If you're planning for this (which you presumably are if you purposefully set the STEPSOUNDS flag or use **footstep_sounds**) you'll have no problem. But this change could very easily break an existing map with the dreaded **ERROR: Index Overflow** error if the number of unique sounds in the map exceeds 256.
2. For single-player games, footstep sounds other than wading use FMOD for playback. The main advantage to using FMOD is that none of those sounds contributes to the 256 sound limit, so you can have as much variety as you want without worrying about Index Overflow. On the other hand, the use of FMOD **requires** the end user to set **s_primary** to 0. Otherwise **all** sounds will be fouled up. Since the game can't possibly know which footstep sounds will be needed at startup (without going through the laborious process of examining the entire bsp), Lazarus precaches all of the footstep sounds at startup using FMOD functions. So we have a chicken-and-the-egg problem here. We want to precache sounds to improve performance, but only if necessary so that we don't force players to change their **s_primary** setting. And how do we know if it's necessary? By checking for STEPSOUNDS and/or **footstep_sounds**. If you stand on your head while reading the previous paragraph it might make more sense.

New surface flags

Apply one of these surface flags to a brush face to play the corresponding footstep sound when the player walks across this surface:

<u>Surface Flag</u>	<u>Hex</u>	<u>Decimal</u>	<u>Description</u>
SURF_METAL	0x00000400	1024	Metal floor
SURF_DIRT	0x00000800	2048	Dirt, sand, rock
SURF_VENT	0x00001000	4096	Ventillation duct
SURF_GRATE	0x00002000	8192	Metal grating
SURF_TILE	0x00004000	16384	Floor tiles
SURF_GRASS	0x00008000	32768	Grass
SURF_SNOW	0x00010000	65536	Snow
SURF_CARPET	0x00020000	131072	Carpet
SURF_FORCE	0x00040000	262144	Force field

There are four sounds associated with each of these surfaces.

Creating *texsurfs.txt*

An example *texsurfs.txt* is included with the footstep sounds distribution on the [Downloads](#) page and is shown in its entirety below. This is a simple ASCII file, each line containing a footstep ID and texture *pattern*. When *footstep_sounds* is set and the surface crossed by the player does **not** have one of the footstep sound surface flags set, Lazarus examines the table read from *texsurfs.txt* and searches for a texture pattern contained in the complete texture name of the surface being crossed.

```
0 /dfloor10_2
0 /dfloor1_4
0 /dfloor3_2
0 /dfloor9_2
1 /c_met
1 /metal
4 /grat
4 /grnx
5 /floor
5 /flr
6 /gras
8 /dfloor
9 field
```

Footstep codes are:

0 Standard footstep	5 Tile
1 Metal	6 Grass
2 Dirt	7 Snow
3 Vent	8 Carpet
4 Grate	9 Force field

The code stops searching the table when it finds a match, so, for example, the first 4 dfloor textures shown in the table (which do not include the red carpet) will play normal footstep sounds, while all other dfloor textures will play carpet sounds.

The number of entries in *texsurfs.txt* is limited to 256 - subsequent entries are ignored. Fellow code geeks are scratching their heads while reading this, thinking that all of these character string comparisons are computationally very expensive. Not to worry... once a match in the table is found, Lazarus forces the appropriate surface flag for this surface on. This switch includes **all** surfaces in the entire map that have the same texture and surface properties, so after finding the first match it is a simple matter of checking surface flags. Hats off to Argh! for pointing out this solution.

WARNING:

For single player games, the code will use FMOD to play footstep sounds, completely bypassing the Q2 sound system. So you can have as many different footstep sounds in a single player map as you'd like, without concerning yourself with Index Overflow (other than the addition of the wading sounds). However, since FMOD doesn't work with multiplayer games, the code will use the normal Q2 sound system for footsteps in deathmatch or coop games. If all of the above sounds are used in the same map, you'll add up to 47 new sounds to the total number of unique sounds. The maximum number of unique sounds played in any one map is 256. If you exceed that limit the game will bomb with **ERROR: Index Overflow**. You can use the [readout](#) cvar to monitor the total number of sounds registered while testing your map. We strongly suggest for deathmatch or coop maps that you **use this feature**.

Construction Notes

Ladders may require a bit of extra work, depending on how you normally construct them. It's interesting to note that up until now, *gamex86.dll* did not know or especially care whether a player was climbing a ladder; ladder physics is handled entirely by the executable. The procedure used by Lazarus to determine whether a player is on a ladder involves projecting the player's bounding box forward by 1 unit and checking the content properties of the first brush intersected by that box. So if your ladder steps use *CONTENTS_LADDER*, but the rails do **not**, this check may fail. To ensure that the code will always correctly determine that the player is on a ladder, either use *CONTENTS_LADDER* for **all** brushes that make up the ladder, or (**better**), surround the ladder with a clip brush that has *CONTENTS_LADDER* set.

WorldCraft users, what can we say? Using new surface flags for footsteps is likely going to cause a bit of frustration for you, since WorldCraft does not allow custom surface flags. To get around this limitation, create the brushes that use custom

footsteps in another editor or in a text editor, then import those brushes into your map in WorldCraft. The **good** news here is that WorldCraft will not delete the new surface flags from your imported brush(es), even if you change other surface flags or content properties of that brush. The corresponding **bad** news is that there is no way to turn these surface flags **off** in WorldCraft. If you want to remove one of the new surface flags, you'll have to delete the brush. Alternatively of course you can ignore all of the above and use texsurfs.txt to specify footstep sounds.

Redistribution

The game will play the 3 stock player/wadeX.wav sounds for wading through water. All others are custom sounds that must be distributed with your map if used.

<u>Surface</u>	<u>Required files</u>
Sloshing	sound/player/pl_slosh1.wav sound/player/pl_slosh2.wav sound/player/pl_slosh3.wav sound/player/pl_slosh4.wav
Ladders	sound/player/pl_ladder1.wav sound/player/pl_ladder2.wav sound/player/pl_ladder3.wav sound/player/pl_ladder4.wav
SURF_METAL	sound/player/pl_metal1.wav sound/player/pl_metal2.wav sound/player/pl_metal3.wav sound/player/pl_metal4.wav
SURF_DIRT	sound/player/pl_dirt1.wav sound/player/pl_dirt2.wav sound/player/pl_dirt3.wav sound/player/pl_dirt4.wav
SURF_VENT	sound/player/pl_duct1.wav sound/player/pl_duct2.wav sound/player/pl_duct3.wav sound/player/pl_duct4.wav
SURF_GRATE	sound/player/pl_grate1.wav sound/player/pl_grate2.wav sound/player/pl_grate3.wav sound/player/pl_grate4.wav
SURF_TILE	sound/player/pl_tile1.wav sound/player/pl_tile2.wav sound/player/pl_tile3.wav sound/player/pl_tile4.wav
SURF_GRASS	sound/player/pl_grass1.wav sound/player/pl_grass2.wav sound/player/pl_grass3.wav sound/player/pl_grass4.wav
SURF_SNOW	sound/player/pl_snow1.wav sound/player/pl_snow2.wav sound/player/pl_snow3.wav sound/player/pl_snow4.wav
SURF_CARPET	sound/player/pl_carpet1.wav sound/player/pl_carpet2.wav sound/player/pl_carpet3.wav sound/player/pl_carpet4.wav
SURF_FORCE	sound/player/pl_force1.wav sound/player/pl_force2.wav sound/player/pl_force3.wav sound/player/pl_force4.wav

Func_bobbingwater

Func_bobbingwater is identical in operation to the Quake2 func_water brush model, except that it bobs up and down in a smooth sinusoidal motion. New keys are **bob** and **duration**. Bob specifies the amplitude of the motion in units; duration specifies the time (in seconds) between cycles. It must be noted that the func_bobbingwater only bobs when triggered; when allowed to "close" on its own, it will not bob during its movement to its original position. If you want the func_bobbingwater to bob at both ends of its motion, then don't set its **wait** to ≥ 0 . Instead, use **wait**=-1 and re-trigger it to "close".

Another feature of the func_bobbingwater (and also the **func_water**) is the ability to give it the properties of mud, which is set, appropriately enough, with the **MUD** spawnflag. This is like water, but different. When the player is in mud, his movement is restricted, and he cannot swim out - he can only walk out. Deep mud means unavoidable sinking. If submerged, the player's view is masked with a dark black-brown view. Drowning in deep mud is a distinct possibility. Movement in mud uses its own set of player sounds.

As outlined in the **redistribution** doc, if the **MUD** spawnflag is set, this entity will require the sound files:

- [sound/mud/mud_in2.wav](#)
- [sound/mud/mud_out1.wav](#)
- [sound/mud/mud_un1.wav](#)
- [sound/mud/wade_mud1.wav](#)
- [sound/mud/wade_mud2.wav](#)

The mud player sound files distributed with Lazarus are courtesy of Maulok (Anthony Bouvette).

Naturally, as with all other moving brush models, transparent textures will not move with the brush when applied to func_bobbingwater. So be sure the transparent texture flag is **not** set.

Key/value pairs

angle

Direction the bobbingwater will move on the XY plane. Default=0.

angles

Direction the bobbingwater will move in 3 dimensions, specified by pitch and yaw. Roll is ignored. Syntax is **pitch yaw 0**. Default=0 0 0.

bob

Specifies the amplitude of motion, in units. Default=16.

duration

Specifies the time between bobbing cycles, in seconds. Default=8.

lip

When activated, the bobbingwater will move [brush thickness]+[-2]-[lip value] units. Default=0.

speed

Speed in units/second that the bobbingwater moves. Default=25.

sounds

Specifies sounds to be played while the bobbingwater is moving. Choices are:

0: silent.

1: water (default).

2: lava.

targetname

Name of the specific bobbingwater.

team

Team name of the specific bobbingwater. Bobbingwaters with a identical team names will move together, and if solid, will all stop if one is blocked. A func_bobbingwater may be teamed with a func_water.

wait

Time in seconds for the bobbingwater to wait before it returns to its "closed" position. If wait=-1, the bobbingwater must be retriggered in order for it to close. This will also result in the **bob** effect taking place on both ends of the bobbingwater's motion, rather than just its "opening" motion. Default=-1.

Spawnflags

START_OPEN Spawnflag (=1)

Bobbingwater will appear in its "open" position rather than its "closed" position.

MUD Spawnflag (=2)

The func_bobbingwater brush will have the properties of mud.

NOT_IN_EASY Spawnflag (=256)

The func_bobbingwater will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_bobbingwater will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_bobbingwater will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_bobbingwater will be inhibited and not appear when deathmatch=1.

Func_button

The **Lazarus** func_button is identical to the standard Quake2 func_button brush model, with the addition of **movewith** support. This allows it to move with its parent entity.

Legend:
Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Direction the button will move on the XY plane. Default=0.

angles

Direction the button will move in 3 dimensions, specified by pitch and yaw. Roll is ignored. Syntax is **pitch yaw 0**. Default=0 0 0.

delay

Specifies the delay in seconds before the button fires. Default=0.

health

When non-zero, makes the button shootable, and specifies hit points required before activating. A shootable button is not pushable. Default=0.

killtarget

Targetname of the entity to be removed from the map when the button fires.

lip

When activated, the button will move [brush thickness]+[-2]-[lip value] units. Default=4.

message

Character string to print to the screen when the button fires.

movewith

Targetname of the parent entity the button is to **movewith**.

pathtarget

If **target** is a trigger_elevator, specifies the path_corner the targeted func_train is to move to.

sounds

Specifies sounds to be played when activated. Choices are 0=audible; 1=silent. Default=0.

speed

Speed in units/second that the button moves. Default=40.

target

Targetname of the entity to be triggered when the button fires.

targetname

Name of the specific button. A button with a targetname cannot be pressed, but it can be shot (if **health**>0) and it can be triggered with another triggering entity.

wait

Time in seconds for the button to wait before it returns to its ready position. If wait=-1, the button will

never return. Default=2.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

Button will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Button will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Button will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Button will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Func_conveyor

The **Lazarus** func_conveyor has a unique feature not found in the original game: *it works!* :-). Func_conveyor may be used to transport just about any entity that is affected by gravity: **func_pushable** crates, monsters, actors, misc_insanes, misc_explobox, gibs, debris, all pickup items, and of course the player. Direction of travel for the conveyor is controlled by the **angle** key; speed is naturally set with the **speed** value.



Construction Tips

You've probably noticed that **all** id conveyor textures are oriented the wrong way to use the FLOWING surface property. (Textures always "flow" from left to right). Rotating the texture doesn't help anything, since the flow direction rotates along with the texture. So the first thing you need for a conveyor that appears to move is a custom conveyor belt texture, rotated 90 degrees from id's standard textures.

Set the FLOWING surface property for your conveyor brushes, but **not** any of the CURRENT content properties. The original func_conveyor called for the use of the CURRENT_ properties; Lazarus' design is more flexible in that conveyors may move at any angle.

You may group all conveyor sections that move at the same yaw angle into a single func_conveyor. In other words one func_conveyor may have multiple sections that have different pitch angles (inclines), as long as they all move along the same yaw angle (angle around the vertical axis). Use the **angle** key to specify the direction of travel for the func_conveyor. Do **not** use "angles" to specify a pitch angle, even for conveyors on an incline. Lazarus determines the correct pitch angle at run time.

Conveyor intersections at the same elevation with different yaw angles are problematic: Objects will almost always get hung up on corners if you attempt this. And really, the conveyor just doesn't look right anyway. You **can**, however, drop objects off the end of one conveyor onto another with no problems. The drop distance is irrelevant: 1 unit is fine. One consideration when dropping objects is the horizontal distance they will travel before landing on the second conveyor. Objects do not slow down in the horizontal direction, so (depending on drop distance and speed) you'll probably want to stop the upper conveyor short of the lower conveyor, or you'll overshoot your target.

As mentioned above, you may use func_conveyor to transport pickup items. There are at least two circumstances in which pickup items should be **SHOOTABLE** if transported by a conveyor:

- If the conveyor doesn't form a closed loop, but instead the map uses some sort of contraption to destroy objects at

the end of the run.

- If the player can access the conveyor (and possibly cause traffic jams). Lazarus is currently merciless with objects that cause pileups on a conveyor, destroying one or all of the objects involved. This may sound drastic, but is quite a bit better than having conveyor traffic backed up. While I'm pretty sure you won't be able to cause pileups with the Lazarus func_conveyor, I've been wrong before :-). If pickups are SHOOTABLE (and thus solid) they are less likely to be involved in visual goofs than if they are nonsolid.

Unlike the original design intent (which cannot possibly work with FLOWING textures), triggering a func_conveyor works much like func_wall: it is either visible, solid, and moves objects... or it is invisible, nonsolid, and moves nothing. To give the illusion of a func_conveyor that is switchable, duplicate the func_conveyor brushes w/o the FLOWING surface property and make those duplicate brushes a func_wall whose initial state is the opposite of the conveyor. Give the func_wall and the func_conveyor the same targetname and set the TOGGLE spawnflag for both.

The base of the bounding boxes of pickup items is generally much larger than the visual model. So when moving up an incline, pickups will appear to float above the surface. You can help hide this problem by constructing a lip on either side of the conveyor that is high enough that the player cannot see the gap below the pickup model.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Direction the conveyor will move on the XY plane. Default=0.

speed

Speed in units/second that the conveyor moves. Default=100. Note that the FLOWING surface property has a set speed of 100 units/second. If you want a faster or slower conveyor that also **appears** to move at the correct speed, you'll have to scale the texture up (faster) or down (slower).

targetname

Name of the specific func_conveyor. If START_ON is not set then the func_conveyor must be triggered to function.

Spawnflags

START_ON Spawnflag (=1)

Func_conveyor will be solid and visible and move objects when the map loads. If START_ON is **not** set, the conveyor will be nonsolid and invisible and will **not** move objects.

TOGGLE Spawnflag (=2)

The func_conveyor will be toggle on/off each time it is triggered

NOT_IN_EASY Spawnflag (=256)

The func_bobbingwater will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_bobbingwater will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_bobbingwater will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_bobbingwater will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Func_door

The **Lazarus** func_door adds **movewith** support, which allows it to move with its parent entity. It also can function as a **movewith parent**, so that other entities may move along with the door. In addition, a negative health value is interpreted as the damage a door will take before being **destroyed**, rather than opened. Debris type produced by destroying the door can be specified with the **gib_type** and **skinnum** values.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Direction the door will move on the XY plane. Default=0.

angles

Direction the door will move in 3 dimensions, specified by pitch and yaw. Roll is ignored. Syntax is **pitch yaw 0**. Default=0 0 0.

accel

Acceleration rate when the door begins moving. Default=0.

decel

Deceleration rate when the door comes to a stop. Default=0.

dmg

Damage in hit points the door will do to an entity that blocks its movement. Default=2.

gib_type

Specifies the debris model to use. Default=0. This value is ignored if **health** is greater than or equal to 0.

0 = Default

1 = Metal

2 = Glass

3 = Barrel

4 = Crate

5 = Rock

6 = Crystal

7 = Mechanical

8 = Wood

9 = Tech

health

When greater than zero, makes the door shootable, and specifies hit points required before activating. If **less than** zero, the absolute value of health specifies hit points required to **destroy** the door rather than open it. In this case the door will "explode" in the same manner as a zero-damage func_explosive, with **mass** taken into account. This feature should not be used with teamed doors. Default=0.

killtarget

Targetname of the entity to be killtargeted when the door is activated.

lip

When activated, the door will move [brush thickness]+[-2]-[lip value] units. Default=8.

mass

Ignored if **health** is 0 or greater. Mass controls the number and size of debris chunks spawned when the door is destroyed. Default=75.

message

Character string to print to the screen when the player approaches the door if its trigger is inactive.

movewith

Targetname of the parent entity the door is to **movewith**.

skinnum

Specifies the skin number to use with the debris models for destroyable doors. As provided with Lazarus, the only debris models that use multiple skins are crate debris (**gib_type**=4). However, all debris models have internal skin references for 8 skins (named "skin0.pcx", "skin1.pcx", ... "skin7.pcx"). So if you want to create your own skins for debris, the code is in place and the model won't require any changes.

0 = box3_7 (explosive crate)

1 = box1_5

2 = box3_1

3 = box3_2

4 = box3_3

sounds

Specifies sounds to be played when activated. Choices are 0=audible; 1=silent. Default=0.

speed

Speed in units/second that the door moves. Default=100.

target

Targetname of the entity to be triggered when the door opens. If the target is a func_areaportal, the areaportal will be triggered again when the door closes.

targetname

Name of the specific door. A door with a targetname cannot be opened without another triggering entity which targets it, but it can be shot (if **health**>0).

team

Team name of the specific door. Doors with a identical team names will move together, and will all stop if one is blocked. With some care concerning timing elements, a func_door may be teamed with a **func_door_rotating**.

wait

Time in seconds for the door to wait before it returns to its closed position. If wait=-1, the door will never return. Default=3.

Spawnflags

START_OPEN Spawnflag (=1)

Door will appear in its open position rather than its closed position.

CRUSHER Spawnflag (=4)

Door will not bounce back open when blocked, and instead will continue to apply its **dmg** value every 0.1 seconds.

NO_MONSTERS Spawnflag (=8)

Door will not be triggered by monsters.

ANIMATED Spawnflag (=16)

Door will display a sequence of animated textures at the rate of 4 animations/second if an animated texture is applied to it.

TOGGLE Spawnflag (=32)

Door must be triggered to close as well as being triggered to open.

ANIMATED_FAST Spawnflag (=64)

Door will display a sequence of animated textures at the rate of 10 animations/second if an animated texture is applied to it.

NOT_IN_EASY Spawnflag (=256)

Door will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Door will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Door will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Door will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Func_door_rot_dh

The **func_door_rot_dh** is a new **Lazarus** brush model, which is identical to the **func_door_rotating**, with one exception: The origin of the brush model is shifted to the origin of the entity specified by the **pathtarget** key at runtime. This feature allows you to build your rotating door in an area where you can ensure that its lighting will be correct, without fouling up the lighting of adjacent world brushes in the process.

Key/value pairs

pathtarget

Targetname of the entity whose origin the **func_door_rot_dh** is to use as a spawning location at runtime. Normally this locating entity should be an **info_notnull**.

For all other keyvalues and spawnflags, please refer to **func_door_rotating** entity, since they are identical.

[Lazarus Main Page](#)

Func_door_rotating

The **Lazarus** func_door_rotating is identical to the standard Quake2 func_door_rotating brush model, with the addition of **movewith** support, which allows it to move with its parent entity, the **turn_rider** key, which allows a player or other entity to rotate with the func_door_rotating if desired, and a change to the definition of the **health** value, which allows you to destroy a door rather than open it by damaging it.

Proper lighting of brush models which incorporate origin brushes is often not easy, since they are unaffected by bounced light, but their surroundings are. While the use of **ArghRad**-specific features can alleviate this effect, **Lazarus** offers another option: the **func_door_rot_dh**.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

distance

Size of the arc in degrees that the door will describe when moving. Default=90.

dmg

Damage in hit points the door will do to an entity that blocks its movement. Default=2.

gib_type

Specifies the debris model to use. Default=0. This value is ignored if **health** is greater than or equal to 0.

0 = Default

1 = Metal

2 = Glass

3 = Barrel

4 = Crate

5 = Rock

6 = Crystal

7 = Mechanical

8 = Wood

9 = Tech

health

When greater than zero, makes the door shootable, and specifies hit points required before activating. If **less than** zero, the absolute value of health specifies hit points required to **destroy** the door rather than open it. In this case the door will "explode" in the same manner as a zero-damage func_explosive, with **mass** taken into account. This feature should not be used with teamed doors. Default=0.

killtarget

Targetname of the entity to be killtargeted when the door is activated.

mass

Ignored if **health** is 0 or greater. Mass controls the number and size of debris chunks spawned when the door is destroyed. Default=75.

message

Character string to print to the screen when the player approaches the door if its trigger is inactive.

movewith

Targetname of the parent entity the door is to **movewith**.

skinnum

Specifies the skin number to use with the debris models for destroyable doors. As provided with Lazarus, the only debris models that use multiple skins are crate debris ([gib_type=4](#)). However, all debris models have internal skin references for 8 skins (named "skin0.pcx", "skin1.pcx", ... "skin7.pcx"). So if you want to create your own skins for debris, the code is in place and the model won't require any changes.

0 = box3_7 (explosive crate)

1 = box1_5

2 = box3_1

3 = box3_2

4 = box3_3

sounds

Specifies sounds to be played when activated. Choices are 0=audible; 1=silent. Default=0.

speed

Speed in degrees/second that the door moves. Default=100.

target

Targetname of the entity to be triggered when the door opens. If the target is a func_areaportal, the areaportal will be triggered again when the door closes.

targetname

Name of the specific door. A door with a targetname cannot be opened without another triggering entity which targets it, but it can be shot (if [health](#)>0).

team

Team name of the specific door. Doors with a identical team names will move together, and will all stop if one is blocked. With some care concerning timing elements, a func_door_rotating may be teamed with a [func_door](#).

turn_rider

If non-zero, a player or other entity riding on the door will rotate as the door rotates. See [this page](#) for details. Default=0.

wait

Time in seconds for the door to wait before it returns to its closed position. If wait=-1, the door will never return. Default=3.

Spawnflags

START_OPEN Spawnflag (=1)

Door will appear in its open position rather than its closed position.

REVERSE Spawnflag (=2)

Reverses the direction of rotation.

CRUSHER Spawnflag (=4)

Door will not bounce back open when blocked, and instead will continue to apply its [dmg](#) value every 0.1 seconds.

NO_MONSTERS Spawnflag (=8)

Door will not be triggered by monsters.

ANIMATED Spawnflag (=16)

Door will display a sequence of animated textures at the rate of 4 animations/second if an animated texture is applied to it.

TOGGLE Spawnflag (=32)

Door must be triggered to close as well as being triggered to open.

X_AXIS Spawnflag (=64)

Door will rotate around the X-axis rather than the Z-axis.

Y_AXIS Spawnflag (=128)

Door will rotate around the Y-axis rather than the Z-axis.

NOT_IN_EASY Spawnflag (=256)

Door will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Door will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Door will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Door will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Func_door_swinging

The **Lazarus** func_door_swinging is a rotating brush model which is similar to a func_door_rotating, in that when it's triggered, it turns. But there the similarities end. This entity can be set up to operate in the following ways:

- Two-way rotating door. It rotates from its initial "at rest" position in both directions, coded to always rotate **away** from its activator. You can build swinging doors and hatches that will never open in the player's face. This is the func_door_swinging's default operation.
- A rotating object that can rotate in either direction repeatedly. Such a door does not "close". When it's triggered open, it remains open. It can be triggered again and it will rotate an additional **distance** degrees again, and will do this in either direction. To enable this behavior, set the **REVOLVING** spawnflag.
- A shootable telltale or similar device. If both **REVOLVING** and **IGNORE_ACTIVATOR** spawnflags are set, and **health**>0, the func_door_swinging will respond as expected depending on what part of the door you shoot.

A plain-vanilla func_door_rotating needs only an origin and an axis to determine its behavior. A func_door_swinging needs an additional bit of information - hence the **followtarget** key. You must point a func_door_swinging's followtarget to an info_notnull, or it will not function. This is so the game knows what side of the door the activator is on, or knows how to bisect the door into "zones" for a shootable **REVOLVING** door. Placement of the info_notnull is easy; if you think of the origin as the door's "hinge", then think of the info_notnull as the "latch". For best results, position the info_notnull *outside* of the func_door_swinging's bounding box.

Automatically-generated trigger fields for rotating doors have always been flaky in Quake2, and this is even more true for the func_door_swinging. For normal 2-way swinging doors, we recommend using a **trigger_multiple** or **trigger_bbox**. Size and position the trigger so that the player (or other activator) cannot trigger the door by approaching it from its side. If this is allowed, the game may be confused as to which side of the door the activator is on, and its opening direction could therefore become unpredictable.

The func_door_swinging also includes the **pathtarget** feature of the **func_door_rot_dh**, which allows for the door to be constructed in one place for lighting purposes, and shifting it to its desired location at runtime.

Key/value pairs

distance

Size of the arc in degrees that the door will describe when moving in one of its two directions. If this value is negative, the door's rotational behavior is reversed. Default=90.

dmg

Damage in hit points the door will do to an entity that blocks its movement. Default=2.

health

When non-zero, makes the door shootable, and specifies hit points required before activating. Default=0.

followtarget

Targetname of the **info_notnull** used to identify the point opposite the door's origin. *All func_door_swinging's must have a followtarget set.* The info_notnull will be removed at runtime for efficiency reasons.

killtarget

Targetname of the entity to be killtargeted when the door is activated.

message

Character string to print to the screen when the player approaches the door if its trigger is inactive.

movewith

Targetname of the parent entity the door is to **movewith**.

pathtarget

Targetname of the entity whose origin the func_door_swinging is to use as a spawning location at runtime. Normally this locating entity should be an **info_notnull**.

sounds

Specifies sounds to be played when activated. Choices are 0=audible; 1=silent. Default=0.

speed

Speed in degrees/second that the door moves. Default=100.

target

Targetname of the entity to be triggered when the door opens. If the target is a func_areaportal, the areaportal will be triggered again when the door closes.

targetname

Name of the specific door. A door with a targetname cannot be opened without another triggering entity which targets it, but it can be shot (if **health**>0).

team

Team name of the specific door. Doors with a identical team names will move together, and will all stop if one is blocked.

turn_rider

If non-zero, a player or other entity riding on the door will rotate as the door rotates. See [this page](#) for details. Default=0.

wait

If **REVOLVING** is not set, this is the time in seconds for the door to wait before returning to its closed position. If wait=-1, the door will never return. For non-REVOLVING doors, the default=3.

If **REVOLVING** is set, this is the time in seconds to wait before the door will accept another trigger input after it completes its move. If wait=-1, the door cannot be triggered again. For REVOLVING doors, the default=0.

Spawnflags

IGNORE_ACTIVATOR Spawnflag (=1)

The code that determines what side of the door the activator is on is disabled.

REVOLVING Spawnflag (=2)

The door will never close, but instead can be repeatedly triggered to rotate **distance** degrees again, in either direction. See also **wait**.

CRUSHER Spawnflag (=4)

Door will not bounce back open when blocked, and instead will continue to apply its **dmg** value every 0.1 seconds.

NO_MONSTERS Spawnflag (=8)

Door will not be triggered by monsters.

ANIMATED Spawnflag (=16)

Door will display a sequence of animated textures at the rate of 4 animations/second if an animated texture is applied to it.

TOGGLE Spawnflag (=32)

Door must be triggered to close as well as being triggered to open.

X_AXIS Spawnflag (=64)

Door will rotate around the X-axis rather than the Z-axis.

Y_AXIS Spawnflag (=128)

Door will rotate around the Y-axis rather than the Z-axis.

NOT_IN_EASY Spawnflag (=256)

Door will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Door will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Door will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Door will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Func_explosive

Lazarus has modified the **func_explosive** brush model so that it can be damaged by impact from other entities, taking mass and velocity into account. Entities with mass less than or equal to 200 (this includes the player) do no damage, regardless of their velocity. For entities with mass>200, the damage is proportional to the square of the impact velocity times the ratio of masses. (For the case where the func_explosive has no mass, 200 is assumed for impact purposes).

- If the striking entity has a mass equal to that of the func_explosive, damage will occur if the impact velocity is greater than approximately 550 units/sec.
- If the striking entity has twice the mass of the func_explosive, damage occurs at approximately 390 units/sec.
- If the striking entity has 4 times the mass of the func_explosive, damage occurs at approximately 275 units/sec.

Lazarus also adds **movewith** support, which allows the func_explosive to move with its parent entity.

The **Lazarus** func_explosive also adds a **delay** key, so that you can build a chain reaction of explosions using func_explosive without risking SZ_GetSpace: Overflow errors. Thanks to our pal Ricebug for the prod to do this.

New **gib_type** and **skinnum** values allow you to specify the use of custom debris models (thanks, *venomus*).

Finally, you may specify that a func_explosive may only be damaged by explosions (rather than impact, projectiles, etc.) with the **EXPLOSIONS_ONLY** spawnflag.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

delay

Specifies the delay, in seconds, after a func_explosive is triggered or killed before it detonates. Default=0. Normally func_eplosive detonates in the same frame that it is triggered or killed, and the resulting damage is also applied to other entities in that same frame. So if you build a long series of func_explosives hoping to create an explosive chain reaction in standard Q2, the normal result will be a game-freezing series of SZ_GetSpace: Overflow errors.

dmg

The amount of damage in hit points that the explosive will do at its origin when triggered. If dmg=0, no explosion effect or sound will occur. Default=0.

gib_type

Specifies the debris model to use. Default=0

- 0 = Default
- 1 = Metal
- 2 = Glass
- 3 = Barrel
- 4 = Crate
- 5 = Rock
- 6 = Crystal
- 7 = Mechanical
- 8 = Wood
- 9 = Tech

health

Specifies hit points before the explosive will fire. Default=100.

killtarget

Targetname of the entity to be removed from the map when the explosive fires.

mass

The amount of debris to spawn when the explosive fires. Also specifies mass used in **impact damage** calculations. Default=75.

movewith

Targetname of the parent entity the explosive is to **movewith**.

skinnum

Specifies the skin number to use with the debris models. As provided with Lazarus, the only debris models that use multiple skins are crate debris (**gib_type**=4). However, all debris models have internal skin references for 8 skins (named "skin0.pcx", "skin1.pcx", ... "skin7.pcx"). So if you want to create your own skins for debris, the code is in place and the model won't require any changes.

0 = box3_7 (explosive crate)

1 = box1_5

2 = box3_1

3 = box3_2

4 = box3_3

target

Targetname of the entity to be triggered when the explosive fires.

targetname

Specific name of the explosive. An explosive with a targetname cannot be damaged by weapon fire or **impact damage**, but instead must be triggered by another triggering entity, UNLESS the **TRIGGER_SPAWN** flag is set.

Spawnflags**TRIGGER_SPAWN Spawnflag (=1)**

Explosive must be called by another entity before it appears in the map.

ANIMATED Spawnflag (=2)

Explosive will display a sequence of animated textures at the rate of 4 animations/second if an animated texture is applied to it.

ANIMATED_FAST Spawnflag (=4)

Explosive will display a sequence of animated textures at the rate of 10 animations/second if an animated texture is applied to it.

EXPLOSIONS_ONLY Spawnflag (=8)

If set, func_explosive may only be damaged by explosions (grenades, rockets, BFG blast, target_explosion, etc.)

Func_force_wall

This brush model entity is taken straight from the Rogue mission pack source code, though we don't recall ever seeing it in use in the Rogue MP. This entity serves largely the same purpose as a func_wall with a force field-type texture, but uses a nifty new particle effect. Contrary to what its name suggests, this is actually a type of trigger field rather than a "wall", so this bmodel is best made using a trigger texture. You can see an example of its use in the **effects** map in the [examples](#).

It can be toggled on and off like a [func_wall](#), but it differs in that if a player or monster is occupying the space where the func_force_wall will appear, he/it will be telefragged rather than merely being stuck. An additional effect is that when pressing up against the func_force_wall, it "gives" rather than appearing completely solid. An optional **dmg** value may be assigned so that an active force wall will do damage when touched.

There are a couple of visual oddities associated with this entity however. Quake2 does not scale particles with distance, so the force wall seems to be much denser when viewed from far away. Also, while the Quake2 engine draws polys based on visdata, it renders particles based on player line-of-sight. So, when suddenly rounding a corner which brings the force wall into view, the particles will appear to have just been turned on at that moment. This entity also seems to have a low priority as far as particle rendering importance goes - if there are a lot of particle effects going on in the vicinity, an active force wall will appear to shut down (but it will still be there).

All that being said, it **is** a nice alternative to the cliched [func_wall](#) force field, and also a viable alternative to using a [func_killbox](#).

Key/value pairs

count

When non-zero, specifies the number of times the force wall will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

dmg

Specifies the amount of health point damage per second that the force wall will do when touched. Default=0.

style

Specifies the color of the force wall particles, using the Quake2 palette index as its definition. For example, 208=yellow, 224=orange, 240=red, 241=blue. Default=208.

targetname

Name of the specific force wall. When triggered, it will toggle on/off.

Spawnflags

START_ON Spawnflag (=1)

Sets the force wall to be active when the map loads.

NOT_IN_EASY Spawnflag (=256)

Force wall will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Force wall will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Force wall will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Force wall will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Func_killbox

The **Lazarus** func_killbox is identical to the standard Quake2 func_killbox brush model, with the addition of **count** support, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

When non-zero, specifies the number of times the killbox will be called before it is auto-killtargeted (see [this page](#) for details). Default=0.

targetname

The name of the specific killbox.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

Killbox will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Killbox will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Killbox will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Killbox will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Func_monitor

The **Lazarus** func_monitor is a new brush model entity which allows you to place security camera viewing stations, robot control stations, and/or **turret** controls in your map. Although func_monitor will work when targeting **any** entity, it is designed primarily to work with either **turret_breach** or **misc_actor**, and the results may be a bit strange if you target any other solid entity: most likely the viewpoint will be embedded in the model for that entity.

The monitor will not work automatically in the same way as a **target_monitor** does. It requires that the player consciously access it with an active **+use** command. In order for this to work the player must be standing within 100 units of the func_monitor and be looking at it. This action will cause the player's viewpoint to shift to the **target** of the func_monitor, and if that target is a turret_breach or misc_actor, he will also take control of its movement. Turning a turret_breach is accomplished with the normal freelook controls. Robot actors may be made to walk or run forward or backward, turn, jump, crouch, and shoot (but **not** strafe). Exiting "monitor mode" is accomplished by another **+use** input.

Misc_actors used as robots are under the complete control of the player. Features:

- Robots do not react to damage and will not attack another entity on their own.
- The robot will only fire at damageable entities. Regardless of the viewpoint offset from the robot, the target selected is the entity that the player is looking at (under the crosshair, in other words).
- As with turrets, if the robot is killed then the player reverts back to normal player control.
- Robots can cause trigger fields that are normally reserved for the player to fire their targets. So robots can open NO_MONSTERS doors and trigger trigger_multiples that do not have the MONSTER spawnflag set. They **cannot**, however, press buttons. So if you want a robot to press a button you'll have to simulate the event by naming the button and targeting the button with a trigger_multiple. Or, of course, you can give the button a health value and the robot can shoot it.
- To help avoid navigation problems, the movement code for the robot maintains a buffer of 16 units between the robot and any solid object. Keep this in mind when constructing any trigger fields that you want the robot to touch.

The func_monitor mechanism works by actually shifting the player to the target and making him invisible and nonsolid. A "fake player" is placed at the monitor where the player was when he accessed it. Monsters will react to this fake player in the same way as the normal player, and if the fake player takes damage, that damage will be assigned to the player himself. If the fake player takes damage while using a monitor, the real player will automatically revert to normal control so he can defend himself. Players cannot die while using a func_monitor, but they can come very close: your health will drop to no lower than 2 points when using func_monitor.

Controlling multiple turrets (or robot actors) from a single func_monitor is covered in the **turret_breach** description. Shifting from one turret/actor to the next while in "monitor mode" is accomplished using the normal movement commands for strafing left and right. Cycling between turrets occurs no faster than once per second.

Key/value pairs

target

Targetname of the entity (or entities) used as the func_monitor's viewpoint. The viewpoint will be offset from the targeted entity's origin by its "move_origin". For a **turret_breach**, this places the viewpoint at the location of the info_notnull used as the muzzle (firing origin) location. The viewpoint will move as the targeted entity moves and/or rotates. If the target is a misc_actor and the misc_actor has no move_origin entry, the viewpoint will be at the *viewheight* of the actor (22 units above the actor's origin)

targetname

Name of the specific func_monitor.

Spawnflags

ANIMATED Spawnflag (=8)

Monitor will display a sequence of animated textures at the rate of 4 animations/second if an animated

texture is applied to it.

ANIMATED_FAST Spawnflag (=16)

Monitor will display a sequence of animated textures at the rate of 10 animations/second if an animated texture is applied to it.

THIRDPERSON Spawnflag (=32)

If **not** set and the monitor's target is a misc_actor, then the misc_actor model will be made invisible while in use. This prevents the player's view from taking on the orange embedded-in-a-solid look. However, this spawnflag has **no** effect on the viewpoint placement, which is determined entirely by the actor's **move_origin** vector (if used).

NOT_IN_EASY Spawnflag (=256)

The func_monitor will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_monitor will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_monitor will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_monitor will be inhibited and not appear when deathmatch=1.

Func_pendulum

The **Lazarus** func_pendulum is... well, a pendulum. It swings back and forth indefinitely, or can be triggered on and off. The motion uses realistic physics: the pendulum accelerates as it rotates earthward, and decelerates to a stop at the top of its arc before accelerating in the opposite direction. As with a real pendulum, the speed of func_pendulum is entirely dependent on its length, specified with the **radius** value. Build the func_pendulum in its at-rest position (for a normal pendulum this is with weight hanging directly below the pivot point). Be sure to add an origin brush at the pivot location, and specify the total arc that the pendulum will rotate through with the **distance** value. By default the pendulum will rotate around the X axis, but you can change this by specifying a non-zero **angle** value. At startup the pendulum will automatically be shifted to its topmost position (pitch angle = distance/2). Set **START_ON** to have the pendulum start rotating immediately, or give it a **targetname** if you would rather trigger it on. See pendulum.bsp in the **examples**.

When blocked, the pendulum will damage whatever is blocking it, provided that the **dmg** value is non-zero and the pendulum velocity is sufficiently high. The **dmg** key specifies the damage to a blocker at the pendulum's maximum velocity (at the bottom of the arc). More on this below. Monsters and players will be knocked completely out of the way by a pendulum, regardless of the monster/player mass or the pendulum **mass**. This is done to prevent the pendulum equations of motion from getting all fouled up. Impact with func_pushables, on the other hand, uses more realistic physics. After-impact conditions in this case are largely determined by the **attenuation** value. If you use a positive attenuation value, the pendulum will slow each time it strikes a func_pushable. How much it slows is dependent on both the attenuation value and the relative masses. While this allows realistic impacts, it also prevents you from building a pendulum slingshot thingamajig to launch multiple func_pushables in sequence, if you're into that sort of thing. In that case you should set attenuation=-1, which will force the pendulum to maintain the same rotational velocity through an impact (unless the func_pushable weighs more than the pendulum, in which case the pendulum will be blocked). Confused? Play around with the pendulum example map and it will come to you.

Mandatory geek paragraph: As with real world pendulums, the period for one complete rotation of func_pendulum is given by $2\pi\sqrt{\text{radius}/g}$, where g is the acceleration of gravity (sv_gravity in Q2-speak). Of note here is that the start angle has no effect on the period. The peak rotational speed of a pendulum is $\text{Theta0} * \sqrt{g/\text{radius}}$, where Theta0 is the pitch angle from vertical at the top of the pendulum's arc. Peak translational speed is then $\text{Theta0} * \sqrt{g * \text{radius}}$ (Theta0 in radians).

Key/value pairs

attenuation

For aspiring engineering students, *attenuation* is the *coefficient of restitution* used in impact calculations when a func_pendulum strikes a **func_pushable**. In plain English, the coefficient of restitution is the ratio of the relative velocities after impact to the relative velocities before impact. In the real world, hard materials tend to have high coefficients, while deformable materials have low coefficients (and so tend to stick together after impact). Valid range is >0.0 to 1.0. Default value is 0.5. This value has **no** effect on player or monster impacts.

If *attenuation* is set to a value less than 0, normal physics is for the most part ignored: if the func_pushable has a greater mass than the pendulum, it blocks the pendulum. If the pendulum has a greater mass, it will maintain its before-impact speed and throw the func_pushable at a speed proportional to the ratio of masses.

distance

Total arc that the pendulum rotates through, in degrees. The maximum distance value is 359 degrees, but really anything greater than 160 or so will look strange if you're attempting to model a realistic pendulum. Default=90.

dmg

Damage to blocker. Actual damage to a blocker will be proportional to the ratio of the current pendulum speed and the maximum pendulum speed (given above). Default=5, unless the maximum speed is less than 200 units/second, in which case the pendulum will do no damage regardless of **dmg** value.

health

Specifies the amount of damage required to destroy the pendulum. When "killed", the pendulum produces a scattering of debris models much as a no-damage func_explosive does. Default = 0 (non-destructible)

mass

Mass is only used when figuring the knockback of solid brush models that block the pendulum motion (e.g. func_pushable). Default = 200.

move_origin

Move_origin is a vector from the origin to the center of gravity of the pendulum. Normally this vector would be (0 0 -radius). However, since you're not limited to building a normal pendulum that swings below a pivot point, **and** because you can fiddle with radius to adjust the speed, the code needs a bit of input to find the location of the pusher when/if something gets pushed.

noise

Sound played upon impact with **func_pushable**. If left blank, defaults to "world/land.wav". The sound will **not** be played if the impact speed is less than 100 units/second.

phase

Specifies a built-in delay for a pendulum, equal to *phase* * period. Phase should be between 0.0 and 1.0. Example - two pendulums that are identical in every way other than phase. One pendulum has phase=0., the other has phase=0.5. If both pendulums are triggered by the same event, the phase=0.5 pendulum will not begin moving until the phase=0 pendulum has reached the opposite topmost position.

radius

The distance from the pivot point to the center of gravity of the pendulum. Default value = 100.

targetname

Name of the pendulum. Func_pendulum without the **START_ON** spawnflag must be triggered (and so must have a targetname) to operate.

Spawnflags

START_ON Spawnflag (=1)

If set, the func_pendulum will begin rotating at startup. Otherwise the pendulum must be triggered to begin moving.

STOP_AT_TOP Spawnflag (=2)

If set, when a moving pendulum is triggered it will continue to its initial topmost position before stopping. If **not** set then under the same conditions the pendulum will stop immediately.

NOT_IN_EASY Spawnflag (=256)

Pendulum will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Pendulum will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Pendulum will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Pendulum will be inhibited and not appear when deathmatch=1.

Func_pushable

The **Lazarus** func_pushable is a brush model which can be moved about by the player or other moving entities in the game. The model can be *any* shape, but the code uses a normal Q2 rectangular bounding box to determine touching, so normal cube-type brushes will be best. A wealth of in-game puzzle-solving possibilities are opened with this entity, especially when used with other entities like the **crane**, the **trigger_inside**, the **trigger_mass**, and **target_attractor**.

Using a func_pushable

Here's how it works: The func_pushable is pushed (or pulled!) by the player walking up against the object and pressing the "+use" key. For pulling, an initial bump in the push direction is required, otherwise when the player backs away, he backs away without the pushable. Push/pull speed is proportional to the ratio of the player's mass to the object's mass. Default mass is 400, which gives a maximum push speed of about 40 units/sec.

It can be a bit difficult for a player to navigate while pushing a large object, so **Lazarus** incorporates a **third-person perspective** which is automatically toggled when the player has control of the pushable. When he releases the object, normal first-person perspective is restored. This can be disabled when running your maps, if desired, by changing the setting for the **tpp_auto** console variable in the version of *default.cfg* included with **Lazarus**.

Destroyable or not destroyable

A func_pushable can take damage and be destroyed in much the same way as a func_explosive can, by giving the pushable a positive **health** value. If you want the pushable to be immune from weapon fire and falling/impact damage (see **below**), then set its health=0 (default). This does not make the pushable invulnerable; it will still be destroyed by closing doors and the like. This prevents the scenario where the player shoves a pushable into a lift shaft, blocking the lift and trapping himself. If you would like the func_pushable to be absolutely invulnerable and block moving brush models, then give its health a negative value.

Gravity and falling damage

This entity functions in a similar way to the func_object where gravity is concerned, but it acts a lot more realistically. Falling uses realistic physics, and the pushable will take damage from the fall. Falling damage is determined in much the same as player falling damage. Use **health<1** to create a func_pushable which does not take falling or impact damage.

It will also do damage to entities that it's dropped on, depending on the **mass** of the pushable and its velocity at the time of impact. This includes players and monsters, as well as other func_pushables, and also **Lazarus**-enhanced func_explosives. If one pushable lands on another with a sufficiently strong impact, both pushables will be damaged, in proportion to the relative masses of the two func_pushables. (Geeky stuff alert): Damage is proportional to the impact velocity squared times the ratio of masses. Damage is given by $\text{mass_p} / \text{mass_m} \times v^2 \times 0.001$, where mass_p=mass of func_pushable, mass_m=mass of monster or player (player mass=200), and v=impact velocity in units/sec.

Note that depending on the animation sequence in use by a monster at the time a func_pushable is dropped on it, the pushable may or may not appear to actually touch the monster. Soldiers, infantry, and tanks are fairly safe bets, but for most other monsters you should probably give the func_pushable sufficient drop height and **mass** to immediately gib the monster rather than simply hurting him. Should the pushable not kill the monster or player it lands on, it will then immediately bounce slightly and fall off to one side - the effect ain't perfect but it's far better than a monster standing there with a crate on his head.

Behavior in liquids

Since **mass** is known, and volume (the size of the brush) is known, determining *density* is the next logical conclusion. (Remember, func_pushables are considered to be parallelepipeds, regardless of their visible shapes, so this is also the basis for determining their volumes). This density value, in turn, can then be used to determine whether a particular func_pushable will float or sink in water, and if it floats, how high it will sit in the water. Quake2 water density is approximately 0.001907 "mass units" per cubic "distance unit", so func_pushables with density>0.001907 (i.e., a 64x64x64 crate with mass=500) will sink. Pushable density of 0.00095 or lighter floats with the waterline at the center of the object. Therefore, to make a realistically-appearing floating object, set the **mass** value of the pushable relative to its size so that the resulting density falls between 0.00095 and 0.001907. Such a pushable will float with the waterline somewhere between its midpoint and its top.

From the above density ranges, you can set how buoyant your floating pushable should be. This requires you to discover the appropriate **mass** value for your pushable, which is easy:

$$[\text{bbox length}] \times [\text{bbox width}] \times [\text{bbox height}] \times [\text{desired density}] = \text{mass}$$

Rider weight is taken into account with floating func_pushables. Little effect would be seen if the player jumps aboard a large floating pushable, but the effect is very noticeable if he gets on smaller ones. In general, floating 64x64x64 crates will ride about 6 units lower in the water if a player stands on it, and the player will usually cause a 32x32x32 crate to sink.

Floating pushables will bob as they float. The less dense the pushable, the higher and faster it will bob. When a func_pushable which can float falls into the water from a height it will dive below the surface as it sheds its downward velocity, and then float back up above its waterline before settling at its floating height. If a pushable is dropped onto another that's already floating, the result will be more or less realistic-appearing physics. The func_pushable already floating will be driven down by the dropped one (and the dropped one may possibly bounce, depending on the relative masses and the impact conditions). Stacked floating func_pushables are... a problem, so we're not allowing them. What happens instead is that the code uses a bit of magic to make stacked func_pushables in water slide off of each other.

Floating pushables will also be carried along by water currents. The more dense the pushable, the more slowly it will move. Maximum speed is 50 units/sec for pushables with density=0.00153 or less (i.e., 64x64x64 crates with mass=400 or lighter). Heavier pushables are moved by currents at a velocity that varies linearly with density.

The player can push and pull a func_pushable that's floating in the water, even if he's swimming in the water too.

Please also refer to the sections at the end of this page concerning **placing pushables in your map** and **limitations**.

Key/value pairs

dmg

The amount of damage in hit points that the pushable will do at its origin when destroyed. If dm=0, no explosion effect or sound will occur. Default=0.

deathtarget

Targetname of the entity to be triggered upon the pushable's destruction.

gib_type

Specifies the debris model to use when func_pushable is destroyed. Default=0

0 = Standard debris models

1 = Metal

2 = Glass

3 = Barrel

4 = Crate

5 = Rock

6 = Crystal

7 = Mechanical

8 = Wood

9 = Tech

health

Specifies hit points before the pushable will be destroyed. If health=0, the pushable is immune to weapon fire and falling/impact damage, but will still explode if it blocks a moving brush model. If health<0 the pushable is completely invulnerable and will block moving brush models indefinitely (also [read this](#)). Default=0.

killtarget

Targetname of the entity to be removed from the map upon the pushable's destruction.

mass

Weight of the func_pushable. The mass value will determine how quickly it can be pushed, if it floats or sinks in a liquid, and how much damage it will do if dropped on another damageable entity. Default=400.

message

Specifies the character string to print to the screen upon the pushable's destruction.

skinnum

Specifies the skin number to use with the debris models for destroyable func_pushable. As provided with Lazarus, the only debris models that use multiple skins are crate debris (**gib_type**=4). However, all debris models have internal skin references for 8 skins (named "skin0.pcx", "skin1.pcx", ... "skin7.pcx"). So if you want to create your own skins for debris, the code is in place and the model won't require any changes.

0 = box3_7 (explosive crate)

1 = box1_5

2 = box3_1

3 = box3_2

4 = box3_3

sounds

Sound to be played when the pushable is moved along while on the ground. Choices are:

0: Silent (default)

1: Tank thud (sound/tank/thud.wav)

2: Railgun hum (sound/weapons/rg_hum.wav)

3: Rocket fly (sound/weapons/rockfly.wav)

targetname

Name of the specific func_pushable. The pushable may be triggered and made to explode like a func_explosive.

Spawnflags

TRIGGER_SPAWN Spawnflag (=2)

The func_pushable must be called by another entity before it appears in the map.

NO_KNOCKBACK Spawnflag (=4)

Normally, damageable func_pushables (**health** > 0) are pushed around by weapon fire in the same way that players and monsters are. Setting NO_KNOCKBACK prevents this behavior.

NOT_IN_EASY Spawnflag (=256)

The func_pushable will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_pushable will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_pushable will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_pushable will be inhibited and not appear when deathmatch=1.

Placing pushables in your map

Quake2 *always* adds a 1-unit buffer to the bounding box coordinates of brush models. In an effort to get around this "feature", **Lazarus** subtracts 1 unit from each side, so that the bounding box matches the model dimensions exactly. This has the nasty side effect of causing adjacent func_pushables (and by adjacent we mean 0 clearance) to be jammed together, and also become inaccessible with the **crane hook**. (In very early versions of **Lazarus**, 2 units were subtracted from each side so as to allow pushables to be stacked adjacent to each other. However this led to problems with func_pushables appearing to be embedded in each other (though they weren't stuck) and a lot of sloppy func_pushable-specific code).

So what does this mean to the mapper? It means this: Use at least a 1 unit gap between adjacent func_pushables in your map. Stacked func_pushables should also have a gap in the z direction. At map load, the func_pushables will "droptofloor" correctly.

Limitations

There are a number of things that keep the effect of this entity from being less than perfect, so keep them in mind:

- Func_pushables can't tumble. So one may be pushed off of a ledge, but looks pretty funky... the object does not fall until fully clear of the ledge. A more realistic motion would be to start tilting once the center of gravity is past the edge of the ledge of course, but Q2's bounding box setup effectively prevents us from doing this.
- The player may *not* move a func_pushable if another is stacked on top of it. The ability to push stacked boxes would allow too many goofy scenarios. For example, what should happen when a crate rests on two other crates, but the center is not above either? Which of the lower crates should it move with?
- Monsters can't push them, since they don't have access to the +use command.
- A func_pushable placed at an angle to the x and y axes makes it just about impossible to push. Keep them square to the grid.
- If it's possible for a func_pushable in your map to be pushed onto a rotating brush model, then the rotating brush model should have its **turn_rider** property set so that the pushable rotates believably. Such a pushable would need an origin brush so that the game knows where its center is; otherwise the map origin (0 0 0) is assumed. Given that the pushable needs to be aligned with the x and y axes (see above) so that it can be manipulated by the player, once it starts spinning about this all goes to hell. Short story is, if the pushable can end up on a rotating bmodel, make sure the player can't try to push it again.
- Indestructible (**health**<0) pushables will block other moving brush models, like closing doors. However the Quake2 code attempts to ensure that indestructible solid objects will never intersect, which means the door will stop when it will intersect on the next game frame, not the current one. In the case of this example with the closing door, this means that the door will actually stop a certain distance away from the pushable. Depending on door speed and the dimensions of the 2 bmodels, this distance can be very noticeable. It will likely take a bit of trial and error to get the brush model speed adjusted such that it will appear to touch (well almost touch anyway) the func_pushable when blocked. The trick is to set the speed such that:

$$[(\text{travel distance}) - (\text{blocker dimension}) + 2] / (\text{blocker speed})$$

...is exactly equal to or slightly less than an integer multiple of 0.1 seconds.

For example, a 128 unit tall door with lip=8 that moves vertically will come to rest on a 64 unit tall func_pushable with speed=97 (0.598 seconds) or speed=116 (0.5 seconds).

[Lazarus Main Page](#)

Func_reflect

The **Lazarus** func_reflect allows you to add mirrors to a map. Umm... well, not **really**, but it provides the illusion of mirrored surfaces. Func_reflect is based on original floor reflection code made public by **psychospaz**. Apologies to psychospaz for my original whining about the limitations of this method - although you'll notice many inconsistencies with the "reflections", if used wisely this feature is pretty darn cool. Hats off to **Argh!** for mercilessly pestering me to include this feature in Lazarus. See the reflect and reflect2 **example maps**.



How it works

In your editor of choice, mirror the world brushes of the room you want reflections in across the reflection plane. We've all seen this technique used before, most recently in the bowling alley of blender81's **Malfunction Junction** and also in Mark Shan's Progetto Genoma. Use e1u1/trigger or other nodraw texture to construct the bounding box of func_reflect. As with trigger brushes, you cannot build complicated shapes with a func_reflect - the extents of the bounding box are the only dimensions that matter. The func_reflect should be on the non-playing side of the reflecting surface and cover the entire area where you want reflections to appear. The **style** setting determines which side of the bounding_box describes the reflecting plane. For example, for style=0 (floor), the top surface of func_reflect is the reflecting plane. At runtime, the code checks for any entity that can be reflected (see below) and finds the location of its reflection for a given func_reflect. If the origin of the reflection would be within the bounding box of that func_reflect, then the entity is reflected. Simple stuff. There are, however, many limitations that you should be aware of:

- The nastiest one: the "reflection" isn't really a reflection at all but a simple rotation. For simple, symmetric entities like ammo and health this is no problem, but for entities that are **not** symmetrical (monsters and the player in particular), if you spend any time at all checking out the reflection you'll notice the inconsistency. The problem with monsters is particularly noticeable with floor reflections. To help obscure the problem with floor reflections you might consider adding an additional transparent layer just below the floor surface. This doesn't fix anything but makes the problem less noticeable.
- Player models are always right-handed, regardless of your *hand* setting. So left-handed players see the weapon held in the correct hand; right-handed players always see the weapon in the wrong hand. There's nothing that can be done about this, other than constructing all new left-handed player models and visual weapons for each new player model... and here's a hint: that won't happen.
- Almost all player models move well outside of their bounding boxes in several animations. You'll likely notice that if

you allow a player to walk up to a wall mirror, the gun of his reflection will protrude through the mirror. The "real" player's weapon also pokes through the mirror, you just never notice that unless you're using **third-person view** or playing deathmatch. We've purposefully left this problem in place in the example map so that you'd notice it (that's our story, anyway). For best results, construct wall mirrors such that the player cannot stand against them.

- The player model lags behind the movement of the real player. This is particularly noticeable when jumping or strafing in front of a wall mirror. Try not to think about it too much :-)
- Entities that normally rotate (weapons, for example) have the rotation turned off automatically by the code. Otherwise the reflection would look exceptionally goofy. But floating non-rotating weapons also look a bit strange, so consider using the NO_DROP and NO_STUPID_SPINNING spawnflags for weapons, as shown in the example.
- All supported entities and effects may be reflected in any or all of the cardinal directions (with a few exceptions noted in the last paragraph). However, only one reflection per direction is allowed. So there's no way to build 2 or more mirrors facing the same direction in the same room but in different planes that all have the correct reflection. You **can** have one func_reflect service 2 or more mirrors that lie in the same plane, as shown in the reflect.bsp example.
- Reflections are not reflected. So it's best to have only one func_reflect per area, despite what you see in the reflect.bsp example map.
- Brush models other than **func_pushable** are **not** reflected. To reflect other brush models, you can construct mirror images of those entities and team them with the "real" entities so that they will move together.
- Func_pushable must include an origin brush, or the reflection will not be in the correct location. The code makes no assumptions about the textures used on the func_pushable, so it is of course best to use a texture that is mostly symmetrical.
- Some effects are reflected, some are reflected only around specific planes, and others are not reflected at all:
 - Rail trails are reflected in all planes, but you'll need to use a bit of care here. Rail trail particles tax the game's particle system heavily - you'll notice in the example map that in many locations, the rail trail will be incomplete in the wall mirrors. This is not a func_reflect bug - it is a limitation on the number of particles present.
 - Explosions other than the BFG are reflected in wall planes but not the floor or ceiling. If you think about it a bit the reason for this is obvious.
 - TE_BFG_LASER, TE_BUBBLETRAIL, TE_BUBBLETRAIL2, and TE_STEAM are reflected in all planes without limitation.
 - Spark-type effects (TE_GUNSHOT, TE_BLOOD, TE_BLASTER, TE_SHOTGUN, TE_SPARKS, TE_SCREEN_SPARKS, TE_SHIELD_SPARKS, TE_BULLET_SPARKS, TE_GREENBLOOD, TE_BLASTER2, TE_MOREBLOOD, TE_HEATBEAM_SPARKS, TE_HEATBEAM_STEAM, TE_CHAINFIST_SMOKE, TE_ELECTRIC_SPARKS, and TE_FLECHETTE) are reflected in walls but not floors or ceilings, since the particles give the appearance of being affected by gravity.
 - TE_FLASHLIGHT is **not** reflected, and you'll notice that it doesn't really behave itself around mirrors. Sorry, there's nothing we can do about this one.
 - Any effect not mentioned above must be reflected manually in the map.

Warning

Simply put: don't overdo it. It is very easy to generate **SZ_GetSpace: Overflow** errors with reflections. In particular, the BFG10K with multiple damageable entities should be avoided.

Key/value pairs

lip

Specifies an additional standoff distance from the applicable func_reflect reflecting plane that reflected entities will be placed. Default value is 3. You'll likely notice that portions of some entities (in particular, the player's feet when looking downward at a floor reflection) tend to penetrate the "mirror" with small lip values. Increasing the lip will solve this problem, but will also cause a noticeable gap between an entity and its floor reflection. So there's a balancing act here. You'll probably find that the default lip

value of 3 is about right for floor reflections.

style

Specifies the side of an area that the func_reflect sits on:

- 0:** Floor
- 1:** Ceiling
- 2:** East wall
- 3:** West wall
- 4:** North wall
- 5:** South wall

targetname

Name of the specific func_reflect. You can toggle the state of func_reflect on/off if the appropriate spawnflags are set, or you might want to killtarget func_reflect after destroying, for example, a breakable func_explosive "mirror". This is particularly well done in the reflect2 [example map](#) - check it out. **Note:** The game will run a bit more efficiently if func_reflect is only operational when a player can see it. So you might want to trigger func_reflect on when a player enters a specific room, and turn it off when the player leaves that same room.

Spawnflags

START_OFF Spawnflag (=1)

If set, the func_reflect will not reflect any entities when the map loads, and must be targeted to work.

TOGGLE Spawnflag (=256)

If set, func_reflect toggles on/off every time it is triggered. If **not** set, func_reflect changes states the first time it is triggered, but subsequent trigger events do nothing.

NOT_IN_EASY Spawnflag (=256)

The func_reflect will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_reflect will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_reflect will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_reflect will be inhibited and not appear when deathmatch=1.

Func_rotating

The **Lazarus** func_rotating is identical to the standard Quake2 func_rotating brush model, with the following changes: The addition of **movewith** support, which allows it to move with its parent entity; the **turn_rider** key, which allows a player or other entity to rotate with the func_rotating, if desired; and the **noise** key, which allows the func_rotating to play a sound only when it's turning.

Proper lighting of brush models which incorporate origin brushes is often not easy, since they are unaffected by bounced light, but their surroundings are. While the use of **ArghRad**-specific features can alleviate this effect, **Lazarus** offers another option: the **func_rotating_dh**.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

dmg

Damage in hit points the func_rotating will do to an entity that blocks its movement. Default=2.

movewith

Targetname of the parent entity the func_rotating is to **movewith**.

noise

Specifies the .wav file to be played when turning. Syntax: [path]/[file].wav.

speed

Speed in degrees/second that the func_rotating turns. Default=100.

targetname

Name of the specific func_rotating. When triggered, it will toggle on/off.

team

Team name of the specific func_rotating. Func_rotatings with a identical team names will turn together, and will all stop if one is blocked. With some care concerning timing elements, a func_rotating may be teamed with a **func_train**.

turn_rider

If non-zero, a player or other entity riding on the func_rotating will rotate as the func_rotating rotates. See [this page](#) for details. Default=0.

Spawnflags

START_ON Spawnflag (=1)

Sets the func_rotating to be active when the map loads.

REVERSE Spawnflag (=2)

Reverses the direction of rotation.

X_AXIS Spawnflag (=4)

Func_rotating will rotate around the X-axis rather than the Z-axis.

Y_AXIS Spawnflag (=8)

Func_rotating will rotate around the Y-axis rather than the Z-axis.

PAIN_ON_TOUCH Spawnflag (=16)

Func_rotating will apply its **dmg** value every 0.1 seconds by simply coming into contact with it - blocking it isn't necessary.

BLOCK_STOPS Spawnflag (=32)

Func_rotating will do no damage when blocked.

ANIMATED Spawnflag (=64)

Func_rotating will display a sequence of animated textures at the rate of 4 animations/second if an animated texture is applied to it.

ANIMATED_FAST Spawnflag (=128)

Func_rotating will display a sequence of animated textures at the rate of 10 animations/second if an animated texture is applied to it.

NOT_IN_EASY Spawnflag (=256)

Func_rotating will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Func_rotating will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Func_rotating will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Func_rotating will be inhibited and not appear when deathmatch=1.

Func_rotating_dh

The **func_rotating_dh** is a new **Lazarus** brush model, which is identical to the **func_rotating**, with one exception: The origin of the brush model is shifted to the origin of the entity specified by the **pathtarget** key at runtime. This feature allows you to build your rotating model in an area where you can ensure that its lighting will be correct, without fouling up the lighting of adjacent world brushes in the process.

Key/value pairs

pathtarget

Targetname of the entity whose origin the **func_rotating_dh** is to use as a spawning location at runtime. Normally this locating entity should be an **info_notnull**.

For all other keyvalues and spawnflags, please refer to **func_rotating** entity, since they are identical.

[Lazarus Main Page](#)

Func_timer

The **Lazarus** func_timer is identical to the standard Quake2 func_timer point entity, with the addition of **count** support, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

When non-zero, specifies the number of times the timer will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

delay

Specifies the delay in seconds before the timer will fire after being triggered.

pausetime

Specifies an initial delay before firing. Similar to **delay**, but only used when the **START_ON** spawnflag is set.

random

Specifies a +/- variance modifier to the **wait** value. The value of random must always be less than the value of wait.

target

The targetname of the entity the timer is to trigger.

targetname

Name of the specific timer. The timer toggles on/off each time it is triggered.

wait

Specifies the time in seconds between firings. Modified by **random**.

Spawnflags

START_ON Spawnflag (=1)

Sets the timer to be active when the map loads.

NOT_IN_EASY Spawnflag (=256)

[ent_name] will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

[ent_name] will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

[ent_name] will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

[ent_name] will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Func_trackchange

Func_trackchange is a moving, rotating platform that transports a **func_tracktrain** from one track to another. It will generally be used after reaching a dead end on the current track, and moving to the start of a new track. When the trackchange arrives at its destination, the train may optionally continue along the new path automatically.

Like standard Q2 func_plats, func_trackchange should be constructed in its top position (defined by **pathtarget**). To build a func_trackchange that starts in its bottom position, set the **STARTBOTTOM** spawnflag. By default, func_trackchange moves along and/or rotates around the Z axis. To create a func_trackchange than moves along and/or rotates around X or Y, set the **X_AXIS** or **Y_AXIS** spawnflags.

Key/value pairs

combattarget

Targetname of the path_track defining the trackchange's "bottom" position.

distance

Angle that the train will turn when travelling from its "top" path_track (defined by **pathtarget**) to its "bottom" path_track (defined by **combattarget**). For non-zero rotations, func_trackchange **must** include an origin brush. Default=0.

height

Travel distance from the "top" (**pathtarget**) to the "bottom" (**combattarget**) path_tracks. Default=0.

pathtarget

Targetname of the path_track defining the trackchange's "top" position.

speed

Maximum translational speed (units/sec) and rotational speed (degrees/sec). Both the translational and rotational speeds are adjusted such that the trackchange will turn through its **distance** rotation and move its **height** distance in the same time. Moreover, the speed is adjusted so that the total travel time is a multiple of Q2's 0.1 second frame time, to avoid overshooting the target and/or slowdowns in the last 0.1 second frame of travel.

target

Targetname of the func_tracktrain moved by the trackchange. If the train is within its **distance** value of the starting path_track, the train will be moved along with trackchange.

targetname

Name of the func_trackchange. This field is required.

Spawnflags

ACTIVATETRAIN Spawnflag (=1)

If set, train will automatically begin moving towards the next path_track when the trackchange reaches its destination. It will move at the same speed it was travelling when the trackchange was activated (if the trackchange was activated by a dead end, the train's speed **before** the dead end was reached will be used).

STARTBOTTOM Spawnflag (=2)

The trackchange will start in its bottom position (defined by **combattarget**).

X_AXIS Spawnflag (=64)

The trackchange will move along and/or rotate around the X axis rather than Z.

Y_AXIS Spawnflag (=128)

The trackchange will move along and/or rotate around the Y axis rather than Z.

NOT_IN_EASY Spawnflag (=256)

The func_train will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_train will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_train will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_train will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Func_tracktrain

The **Lazarus** func_tracktrain brush model provides the same sort of functionality as its namesake from Half-Life. It moves along a path defined by a series of **path_track** entities, and gives the player control over speed and direction. Unlike its HL counterpart, the **Lazarus** func_tracktrain has no problems following a path that doubles back upon itself. This means you can include reverse loops, wyes, and use **func_trackchanges** as turntables in your map, and the func_tracktrain will return to follow the same set of path_tracks in the reverse order that it used them to get there.

Using a func_tracktrain: Func_tracktrain is engaged by the player by pressing +use when the player's origin is within the defined control area (see **construction notes** for details). The player loses control of the train when jumping or strafing, by pressing +use again, or when he is killed. When in control of the tracktrain, the player will see a speed/direction indicator on the screen (unless **NO_HUD** is set). Optionally, the speed controls can appear on the tracktrain itself if a custom animated texture and the **ANIM_INDICATOR** spawnflag is set. He can change speed/direction by using the normal forward/back movement controls. While driving, he can look around but cannot move relative to the tracktrain; if he disengages, the tracktrain will continue at its last speed/direction setting and the player may continue to ride it if he wishes.

Level changes: You can drive/ride a tracktrain from one map to the next and back again, so you need not limit your track network to a single map. For the scoop on how to get this working, read the **notes** at the end of this page.

Movewith: Func_tracktrain can function as a **movewith** parent, so that other entities can be made to move along with the train.

Monster drivers: By placing a **DRIVE_TRAIN** point_combat at the driving position for the train, you can force monsters to drive func_tracktrain. A monster-driven train will always travel at full speed and immediately reverse directions at dead ends. Additionally, if the train is more than 2000 units away from the monster enemy's last known position, **and** the monster has seen his enemy since the last direction change, the train will reverse direction. Since the train might be moved to an unpredictable spot before the monster is triggered to find the point_combat, it's a good idea to set the **movewith** value of the point_combat to the targetname of the func_tracktrain.

As outlined in the **redistribution** doc, unless **NO_HUD** is set, this entity will require the following files:

- pics/speed0.pcx
- pics/speed1.pcx
- pics/speed2.pcx
- pics/speed3.pcx
- pics/speedr1.pcx
- pics/speedr2.pcx
- pics/speedr3.pcx

In addition, unless **sounds** is less than 0 (for a silent train), these files will be required:

- sound/train/X/speed1.wav
- sound/train/X/speed2.wav
- sound/train/X/speed3.wav

where *X* is the sounds value (1 if not specified).

Many options are available for customizing your func_tracktrain, as outlined in the keyvalue/spawnflag descriptions below. Additionally, please read the sections at the end of this page concerning **construction**, **movement**, **level changes**, **train feedback**, and **miscellaneous info**.

Key/value pairs

bleft

Specifies bottom-left (minY/minX/minZ) bounding box coordinates for the train control field, using the tracktrain's origin as the reference. See also **tright**. Ignored if **NOCONTROL** is set.

Default=(-16 -16 -16).

distance

This may loosely be defined as the train wheelbase. The train constantly looks ahead along its path by this value to determine yaw and pitch angles. Larger values may result in smoother rotations. Suggested value is the length of the train. Default=50.

dmg

Specifies the damage inflicted by the func_tracktrain every 0.1 seconds on entities that block its progress. Default=0.

height

Sets the height above path_tracks that the train will ride. Unlike func_train, func_tracktrain is placed such that its origin (rather than its minimum bounding box coordinates) is the reference point for placement. Default=4.

message

Character string to print to the screen the first time the player enters the **bleft/tright** train control zone of an enabled train. This message will only appear once. Please also see **train feedback** notes. Ignored if **NOCONTROL** is set.

roll

Sets the bank angle when going through turns at any speed. Non-zero values will override **roll_speed**. See **movement** notes for more details. Default=0.

roll_speed

Similar to **roll**, this sets the bank angle when going through turns at full speed. The roll_speed value actually used is in a 1:1 ratio with **speed**, so at 1/3 speed the roll angle will also be 1/3. If **speed** is changed by a path_track speed multiplier, roll_speed will be altered as well by the same factor. Ignored when **roll** is non-zero. See **movement** notes for more details. Default=0.

sounds

Defines the family of sounds used by the train:

sounds description

- 1 none (silent)
- 1 steel wheels
- 2 squeaky steel wheels

Default = 1. Future releases will include additional sounds in each family, and most likely one or two new sound families.

speed

Sets the maximum speed of the tracktrain. Func_tracktrain has 3 forward speeds and 3 reverse speeds, each in increments of speed/3. This value may be changed in-game at any **path_track**. Default=64.

target

Targetname of the path_track where the func_tracktrain will spawn. The train will be oriented such that it faces the target of this path_track. (Note that the train should be constructed so that its front faces east in the map editor).

targetname

Name of the specific func_tracktrain. Each time the tracktrain is triggered it will toggle between a disabled and enabled state, unless **NOCONTROL** is set, in which case it would be toggled on and off.

tright

Specifies top-right (maxY/maxX/maxZ) bounding box coordinates for the train control field, using the tracktrain's origin as the reference. See also **bleft**. Ignored if **NOCONTROL** is set. Default=(16 16 16).

viewmessage

Character string to print to the screen when the player enters the **bleft/tright** train control zone and the tracktrain has been disabled by calling its **targetname**. Please also see **train feedback** notes. Ignored if

NOCONTROL is set.

Spawnflags

NOPITCH Spawnflag (=1)

The func_tracktrain will not pitch as it changes grades.

NOCONTROL Spawnflag (=2)

The func_tracktrain cannot be controlled by the player in the normal manner, and instead must be triggered directly by calling its **targetname**. When activated, it immediately accelerates to its full **speed**.

ONEWAY Spawnflag (=4)

The func_tracktrain may move forward only, and can never back up.

OTHER_MAP Spawnflag (=8)

The func_tracktrain will start out invisible, nonsolid, and disabled. Such a tracktrain cannot be used until an active tracktrain with matching targetname from another map is "moved" to this tracktrain's map via **trigger_transition/target_changelevel**. When this spawnflag is set, all other keys and flags are ignored, since OTHER_MAP tracktrains inherit their properties from the tracktrains they replace. See **level changes** notes for details.

NO_HUD Spawnflag (=16)

The func_tracktrain will not cause the HUD graphic to appear when the player takes control. Please also see **train feedback** notes.

ANIM_INDICATOR Spawnflag (=32)

The func_tracktrain will display selected frames of a 15-frame (0-14) animated texture, if applied, which can serve to indicate the tracktrain's speed/direction status. Please also see **train feedback** notes.

<u>Train Status</u>	<u>Texture Frames Displayed</u>
Full reverse	frame 0 and 1
2/3 reverse	frame 2 and 3
1/3 reverse	frame 4 and 5
Disabled	frame 6 (not animated)
Stopped	frame 6 and 7
1/3 forward	frame 8 and 9
2/3 forward	frame 10 and 11
Full forward	frame 12 and 13
Disabled	frame 14 (not animated)

Note: if **ANIMATED** is set in conjunction with this spawnflag, the result is ANIMATED_FAST (10 frames/sec).

ANIMATED Spawnflag (=64)

The func_tracktrain will display a sequence of animated textures at the rate of 4 animations/second if an animated texture is applied to it. Animations are only displayed when the tracktrain is moving.

Note: if **ANIM_INDICATOR** is set in conjunction with this spawnflag, the result is ANIMATED_FAST (10 frames/sec).

START_OFF Spawnflag (=128)

The func_tracktrain will be disabled when the map loads. This spawnflag applies to drivable trains only. Drivable trains toggle between disabled/enabled each time they are triggered.

NOT_IN_EASY Spawnflag (=256)

The func_tracktrain will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_tracktrain will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_tracktrain will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_tracktrain will be inhibited and not appear when deathmatch=1.

Construction

Since a func_tracktrain is a rotating entity, it needs an origin brush as part of its construction. It must be designed so its front faces angle=0 (east); when the map loads in the game, the tracktrain will spawn at its **targeted** path_track, and will be facing next path_track defined by the first path_track's target. Train controls are built in; there's no need for a separate train control entity. The "active site" of the control area is within the bounds of an imaginary box defined by **bleft** and **tright**. The player can take control of the tracktrain when his origin is within this zone. Be sure to define this area carefully for tracktrains that will make grade changes, since the player's bounding box will not pitch as the tracktrain does. The vertical planes of the player's bounding box should never intersect any surfaces of the tracktrain as it pitches.

Movement

The best way to make a smooth-running tracktrain is with lots of intermediate path_tracks along its route where it makes turns and changes grades. However the tracktrain's **distance** value can greatly affect how realistically the tracktrain will move. In general, fast trains on tracks with sweeping turns will benefit from large distance values, while slow trains on tight turns would need smaller distance settings.

When coupled with the placement of the tracktrain's origin, **roll** or **roll_speed** (you can use one or the other, but not both) can give a nice illusion of speed and momentum around turns if not overdone. "Roll" is useful for banked turns, while "roll_speed" is designed to model centrifugal force, and will change in proportion with tracktrain speed changes. Remember that a func_tracktrain will roll around its origin, not around its path_track path. Positive roll/roll_speed values cause the area above the tracktrain's origin to roll to the outside of a turn; negative values roll to the inside of the turn. So if you wanted to make a swinging skycar for example, construct the tracktrain so that its origin is above the car (ideally, it should be at the same level as the "cable" that "suspends" it), and give it a negative roll_speed value, and you'll end up with a skycar that swings outward on turns as you would expect. Whatever design you make, try and make it move logically.

Level changes

Lazarus func_tracktrain can travel from one map to the next, or more precisely it gives the illusion of doing so. Brush models cannot change levels, so a new form of func_tracktrain is needed: an **OTHER_MAP** tracktrain. OTHER_MAP tracktrains are kind of "dummy" tracktrains that don't appear in the game unless they're needed for their brush model. If you want a tracktrain to change levels, make the func_tracktrain that will appear in the first map, and give it all the keys and spawnflag settings you want it to have. Then copy that tracktrain and place a copy in each of the maps it may potentially appear in. Set the **OTHER_MAP** spawnflag for all the copies; these copies will not use any of their own keyvalue/flag settings, but will instead inherit the properties of the original tracktrain, should it move to one of those maps. If the original tracktrain doesn't change levels, the "dummy" tracktrains won't appear.

Now, here's how it works: If the func_tracktrain origin is within the bounds of a **trigger_transition** with targetname matching that of a triggered **target_changelevel**, that tracktrain is made nonsolid and invisible. In the next map, an **OTHER_MAP** tracktrain with matching **targetname** is placed in the appropriate location, with the same velocity, angles and all other properties of the original tracktrain. If the player was driving the tracktrain when the level change occurred, he'll continue to drive the "new" train in the next map. The process repeats for subsequent level changes, and the tracktrain will never appear to exist in multiple maps at the same time.

Note that this necessitates the duplication of **path_track** paths in both maps in a given transition area. Path_tracks do **not** change maps, so they must be duplicated - and this includes giving those path_tracks in the adjacent maps **identical** keys and spawnflags. This is because information of where a func_tracktrain is, where it's going, and where it's been is saved by the tracktrain itself. If its new path_track environment does not match its old one, the tracktrain will become, at best, confused. The reason we don't permit path_tracks to change levels is because if we did, you could go from map A to map B, taking the path_tracks with you, then return to map A at a different point... and then map A would be missing a few path_tracks.

Train feedback

When a player encounters a func_tracktrain in your map, he needs to know what he's found, and how to use it.

Probably everyone reading this has played Half-Life, but it's not a valid assumption that everyone who's downloaded you map to play has played Half-Life, and even if they have, there's really no reason for them to expect to encounter a tracktrain in Quake2.

As first-person-shooter games have evolved from games where the player merely shoots stuff and bumps into clearly-marked buttons, and have begun to include entities the player can interact with, the need for pre-game tutorial maps was born. In these maps, the player is introduced to the skills he would need to later get through certain sequences... one of those skills is operating tracktrains. The question is, does the mapper who's linking together 2 or 3 maps in a unit want to lead that unit off with a tutorial map? We think he doesn't, nor do we think a player wants to slog through a tutorial map for every handful of maps he plays.

We believe (and you're certainly free to think otherwise) that the function provided in such tutorials can be integrated into the maps that use such entities (like tracktrains). You could try ignoring the problem entirely, and hope the player figures it out, or that he's played the same games you have and knows the drill. Do that at your own peril. Instruction in an attached readme file doesn't cut it either. The conclusion here is that the mapper has to provide some in-game instruction, in such a way that it's informative and at the same time not annoying. This is not an easy thing to do, since there's a fine line between beating the player over the head with instructions and not giving enough information.

The **Lazarus** `func_tracktrain` provides some tools for you to clue the player in on what's happening. The `message` key lets you tell the player what to do when he's in the right place to operate the train for the first time. Here's where you can tell the player to press the `+use` key perhaps. The `viewmessage` key is where you tell the player why the tracktrain isn't working, so as to prevent him from cursing in frustration. By themselves these message strings are probably not quite enough, but if the tracktrain is presented in your map in a logical context, things hopefully will work out OK gameplay-wise.

Once the "what is this thing and how do I use it" hurdle is behind you, the next thing needed is in-game feedback from the tracktrain that tells the player if he's stopped or moving, which way he's going, and how fast. By default a Half-Life-like HUD graphic is displayed for this purpose. You may feel this is crossing the line into beating the player over the head, after all the player can see if he's moving or not and may also understand his speed from the audible train sounds. Which is why we provide the `NO_HUD` option. Or perhaps you want some sort of indicator, but you don't want the in-your-face HUD graphic; in that case you might want to put the speed/direction indicator on the tracktrain itself - that's where the `ANIM_INDICATOR` spawnflag comes in.

However your map and tracktrain design turns out, the final test lies in the hands of your playtesters. Try to find a playtester who's never played a game with a tracktrain-type entity in it. If he can figure things out then that's a good sign. But also have a savvy player who's familiar with Half-Life (or even **Lazarus**) tracktrains play your map too - if your tracktrain instructions don't annoy him then you know you haven't gone overboard with them. If you can make both these guys happy you've done a damned good job. Remember, frustration and annoyance are not synonyms for fun, and above all your map is supposed to be fun.

Miscellaneous Info

The information presented here doesn't fit nicely into any of the above categories, but you might find some of this useful in specific situations:

- Monsters will never dismount a train once they take control, so if a player wants to drive the train, he must kill the monster. That sounds easy enough, but depending on the available weapon selection, train speed, and monster health, may be a bit tough to do. Consider adding a trap to knock off the monster as is done with the explosive `func_pushable` in the *track1* example. One change has been made to the radius damage code to facilitate this a bit: `Func_tracktrain` drivers are not shielded from an explosion by the train they are driving.
- "On a Rail"-type train wrecks can wreak havoc on the driver's health if you don't take a few precautions. In the *track1* example, a `target_attractor` is used to give the driver a small boost, lifting him away from the train. The effect gives the appearance of the driver being catapulted from the train as it pitches downward, so it isn't a bad thing. If this were **not** done, the combination of a sudden increase in speed, sudden pitch change, and of course the earthquake would combine to make it very likely that the driver would become embedded in the train at the moment it begins its "wreck".
- In the same scenario as above, once the train comes to rest (at a significant pitch angle) and the former driver lands on the train, depending on train construction you might need a few clip brushes to prevent the player/monster from becoming stuck in and/or hurt by the train. This is not a `func_tracktrain` problem, rather it is a problem with angled brushes in general. We've all seen cases of taking damage in Q2 maps simply by

standing on brushes at steep pitch angles. In the *track1* example map a couple of clip brush func_walls are used to prevent the former driver from becoming embedded in the train.

Lazarus Main Page

Func_train

Lazarus allows you to change the speed of a func_train at every **path_corner**, rotate a func_train using one of three methods, and destroy a func_train. It also provides a solution to all your func_train sound worries. Func_train can serve as the base platform for a variety of other entities by using the **movewith** key with those entities.

Legend:
Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

dmg

The damage inflicted by the func_train on entities that block its progress. Ignored if **BLOCK_STOPS** is set.

health

A non-zero health value will make your func_train damageable by weapons fire. When "dead", the func_train will explode with a zero-damage explosion.

noise

If you've ever tried using the "noise" value for a func_train, you've either used an origin brush with the func_train, placed the func_train at 0,0,0 in the map, or wondered why the sound didn't play. If the noise value is used, the Lazarus game dll creates a speaker entity that follows the center of the brush model around, playing the sound specified by "noise" when the entity is moving. This eliminates the need for an origin brush with a func_train and the accompanying lighting problems, and also allows the use of a sound for a toggled func_train (sounds played with "normal" func_trains can only be turned on and off at path_corners).

Alternatively, if the train moves a large distance and the normal train sound attenuates more rapidly than you'd like, you can tie a 3D **target_playback** sound to the motion of the train with its movewith value.

pitch_speed

roll_speed

yaw_speed

For **ROTATE** trains, pitch_speed and yaw_speed are the rotational velocities of the train while turning towards the next **path_corner**. Roll_speed is the rotational speed while rolling to the next path_corner's **roll** value. Once the train faces the next path_corner, yaw and pitch rotations stop. Likewise, once the train's roll angle is equal to the next path_corner's roll value, roll rotation stops. There are no default values for these speeds, so if you want the func_train to turn towards a path_corner and/or roll, you **must** supply non-zero values for the corresponding rotational speeds. Conversely, if you want your func_train to change yaw but not pitch for path_corners at different elevations, set pitch_speed to 0.

For a **ROTATE_CONSTANT** train, these are the constant rotational values that will be used by the func_train until either the train is toggled off or a path_corner changes the value. As with ROTATE trains, these speeds may be changed at every path_corner. **However**, for ROTATE_CONSTANT trains the path_corner values have a different meaning. Rather than specifying absolute values, the path_corner values are the **change** in speed. This was done so that 0 or unspecified values at path_corners have no effect, but you can still stop rotation by using the negative of the current rotational speed.

Finally, for a **SPLINE** train, roll_speed is never used, and pitch_speed and yaw_speed are **only** used to prevent rotation in a given direction by setting to a value less than 0. For example, if you don't want a train to change its pitch angle even though the path_corners comprising the train's route are at different elevations, set pitch_speed=-1.

speed

No changes to the func_train itself, but speed may now be changed at every path_corner to the path_corner speed value. If the path_corner speed value is 0 or not specified, the train speed does not change.

target

Targetname of the first path_corner on the func_train's route.

targetname

Name of the func_train. Unnamed func_trains will always start on, regardless of the START_ON setting.

team

Team name of the func_train. Trains with an identical team name will move together, and will all stop if one is blocked.

turn_rider

If non-zero, all gravity-affected riders will be turned along with the train. You've probably noticed that in the standard game, a player riding a func_rotating does **not** turn with the func_rotating. Lazarus requires a new key to override this behavior so that existing maps won't be broken. The turn_rider value also has a special meaning when applied to a func_train with movewith children: If non-zero, train rotation will be applied to the child. If zero, the child rotation will be independent of the train.

Spawnflags

START_ON Spawnflag (=1)

By default a train with a targetname will only start moving when triggered by another entity. If you set the START_ON spawnflag then the func_train will start moving when the level starts.

TOGGLE Spawnflag (=2)

If set, allows the train to be toggled on/off multiple times.

BLOCK_STOPS Spawnflag (=4)

The func_train will stop completely when an entity is blocking its way. The blocking entity will not take damage, even if a positive dmg value is used.

ROTATE Spawnflag (=8)

Thanks to Rroff for providing the basis for rotating train code. If set, func_train will alter its pitch and yaw angles to turn towards the next path_corner in its sequence, and also alter its roll angle to turn towards the roll value of that same path_corner, unless that path_corner has the NO_ROTATE spawnflag set. Rotational speeds are controlled with the pitch_speed, roll_speed, and yaw_speed values.

Combining ROTATE and ROTATE_CONSTANT internally sets the SPLINE spawnflag and turns those flags off.

ROTATE_CONSTANT Spawnflag (=16)

If set, the func_train will constantly rotate around one or more axes continuously, until the train is toggled off or the rotational speeds are changed by a path_corner.

Combining ROTATE and ROTATE_CONSTANT internally sets the SPLINE spawnflag and turns those flags off.

ANIMATED Spawnflag (=32)

Will cycle through texture animations while train is on, 1 frame/second.

ANIMATED_FAST Spawnflag (=64)

Will cycle through ALL texture animation frames in 1 second while train is on.

SMOOTH_MOVE Spawnflag (=128)

You've probably noticed in standard Quake2 that func_trains tend to stutter a bit at path_corners. For all of you gearheads out there, here's why: Q2 divides the distance between path_corners by the train speed to get the travel time, then truncates that time to the nearest 0.1 seconds. The func_train is then

sent on its way to the path_corner, and the game won't worry about the train again until that time has expired. At the end of that time, the game finds the remaining distance to the path_corner. If the train has not been blocked, this distance will always be between 0 and 0.1*speed. Now... here's where the stutter comes in. Quake 2 entities other than players only "think" every 0.1 seconds. It would be meaningless to tell the train to travel at its normal speed for a time less than 0.1 seconds then stop... it just can't do that. So to get the train to its intended destination, the game slows the train down so that it will arrive at the next path_corner 0.1 seconds later. If the remaining distance had been **exactly** 0, then this slowdown would not occur. But if the path between path_corners is not parallel to x, y, or z, small roundoff errors will almost certainly cause the train to use that last 0.1 second slowdown. Well... this is all well and good (and necessary) for precise train placement, but very often smooth movement (constant velocity) is more important than precise placement. This is particularly true of simulated vehicles. Enter SMOOTH_MOVE. If set, the game will skip the last 0.1 second slowdown and proceed as if the train had arrived at the path_corner, though it may be up to 0.1*speed units short of that location. This does **not** effect path_corner pathtargets in any way - they will still be triggered. However, you should **not** use this spawnflag if you are relying on precise train placement (say to crush an object, for example).

SMOOTH_MOVE is meaningless with a **SPLINE** train, as the game internally adjusts the train speed to give the same effect.

NOT_IN_EASY Spawnflag (=256)

The func_train will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_train will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_train will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_train will be inhibited and not appear when deathmatch=1.

SPLINE Spawnflag (=4096)

Big kudos to our man **Argh!** for providing spline fit code for use in Lazarus. If SPLINE is set, the func_train will follow a spline curve between path_corners. What this means is you can create near perfectly smooth curvilinear paths with a handful of path_corners. The train will constantly turn to face the direction it is moving (unless **yaw_speed** and **pitch_speed** are negative values). For a couple of examples, see the rottrain and lcraft example maps (available on the **downloads** page.)

For WorldCraft (which doesn't allow spawnflags > 2048) users, a combination of the **ROTATE** and **ROTATE_CONSTANT** spawnflags is translated internally to SPLINE (and ROTATE and ROTATE_CONSTANT are turned off).

The shape of the spline curve is controlled by the location and pitch and yaw angles of the train's **path_corners**. Train roll angle will vary linearly between path_corner roll angle values (the third component of the angles vector).

Func_trainbutton

The **Lazarus** func_trainbutton is a new brush model entity which is based on the standard Quake2 func_button, except that it can move along with a func_train. The parent train is specified with the **movewith** key.

NOTE: Func_trainbutton is obsolete for versions 1.2 and later, since support for the **movewith** value has been expanded and improved. You can now use a normal **func_button**. We'll keep support for func_trainbutton in place so that we don't break any existing maps which happen to use it.

For all keyvalues and spawnflags, please refer to the **func_button** entity, since they are identical to that of the func_trainbutton.

[Lazarus Main Page](#)

Func_vehicle

The **Lazarus** func_vehicle is a brush model that may be "driven" by the player. It has 3 forward speeds and 3 reverse speeds, and can be steered (it does not follow a predetermined path). For the player to operate the vehicle, he mounts it and presses the **+use** key once (the player needs to be within 16 units of the **move_origin** for the vehicle to recognize this). Dismounting requires pressing **+use** a second time. Driving it forwards and backwards is done with the normal player movement forward and back commands; normal player strafing commands cause the vehicle to turn (the vehicle must be moving for turning to take place). The player will turn while the vehicle turns. While driving, the player can use freelook to look around.

When the player is in control of a func_vehicle, but the vehicle is not moving, an idle sound is played. When the vehicle is moving, another sound is played. A pair of custom engine sounds are included for this purpose, which may be replaced at the mapper's discretion. Likewise, a group of image files which look hauntingly like Half-Life's func_tracktrain direction/speed indicator are included.

Because the func_vehicle turns, it needs an origin to rotate around. Either make an origin brush part of the vehicle, or place the vehicle so that its desired origin is at the map coordinate 0 0 0 (which requires making the map with the vehicle's desired position in mind).

Func_vehicle is clipped to the world... in other words it will not travel through world brushes. Collision detection will work best if the origin is placed at the center of the func_vehicle (in X and Y; Z isn't important for collision detection, but it **is** important to set the proper sound origin).

The vehicle can do collision damage to other entities, and in turn can sustain damage to itself. Impact damage is proportional to the **mass** of the vehicle times the impact velocity squared. If the vehicle's **speed** is less than 200 units/sec, it will push monsters, actors, and players rather than damage them. The vehicle will not stop or lose speed when destroying obstacles, which may not be entirely realistic, but it does allow for plowing through multiple func_explosive barriers, for example.

A func_vehicle may function as a **movewith** parent, so that other entities may be attached to it. To support this, func_vehicle can make use of the **turn_rider** key. But unlike "turn_rider" use with other rotating bmodels, this doesn't determine whether or not the player turns with the vehicle, since he always will - it only determines whether or not movewith children will turn with the vehicle.

As outlined in the **redistribution** doc, this entity will require the following files:

- pics/speed0.pcx
- pics/speed1.pcx
- pics/speed2.pcx
- pics/speed3.pcx
- pics/speedr1.pcx
- pics/speedr2.pcx
- pics/speedr3.pcx
- sound/engine/engine.wav
- sound/engine/idle.wav

Also refer to these **notes** regarding construction tips, limitations, and other details you should be aware of.

Key/value pairs

angle

Specifies the facing angle of the vehicle on the XY plane. The func_vehicle should be constructed so that its intended "front" faces 0; the value of angle determines the direction the front will face when the map is loaded. Default=0.

dmg

Damage in hit points the vehicle will do when destroyed, accompanied by the familiar fireball. If dmg=0, no fireball will appear. Default=0.

deathtarget

Targetname of the entity to be triggered when vehicle damage points meets or exceeds its **health**.

health

Damage hit points the vehicle will sustain before blowing up. If health=0; the vehicle is invulnerable.

mass

Weight of the vehicle. Used for calculating impact damage to other entities (destroyable bmodels, monsters, other players, etc). Default=2000.

message

Text string displayed to the screen when the player takes control of the func_vehicle for the first time.

move_origin

Specifies the offset in 3 dimensions from the func_vehicle origin to the driver origin. The player origin must be within 16 units of this point before the game allows him to take control of the vehicle. Default=0 0 0.

radius

Specifies the turning radius of the vehicle in map units. Default=256.

speed

Maximum allowable full forward speed of the func_vehicle, in units/sec. Func_vehicle accelerates to full speed in roughly 1 second. It decelerates by 1/2 of its current speed every frame, until the speed is at or below 10 units/sec (at which point it comes to a full stop). Default=200.

targetname

Name of the specific func_vehicle.

turn_rider

If non-zero, and the func_vehicle is serving as a **movewith** parent, then its movewith children will rotate as the func_vehicle turns. See [this page](#) for details. Default=0.

Spawnflags

BLOCK_STOPS Spawnflag (=4)

The func_vehicle's movement will be blocked by all solid entities, including monsters, players, func_explosives and func_pushables.

NOT_IN_EASY Spawnflag (=256)

The func_vehicle will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_vehicle will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_vehicle will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_vehicle will be inhibited and not appear when deathmatch=1.

Notes

- Func_vehicle will not climb or descend; it may only travel on a level surface.

- The code to determines the "front" of the vehicle to be the part of the its brush model that faces 0 degrees. Therefore it should be constructed so that the end you intend to be its "front" faces that way. The vehicle's in-game facing direction can be anything you wish... this is set with the **angle** key. If you've ever constructed a turret_breach, this procedure will be familiar to you.
- Because of the way rotating entities in Quake2 are represented as rectangular bounding boxes, it hasn't been possible (so far) to completely eliminate the possibility of func_vehicles intersecting world brushes as they turn. For example, if the vehicle is longer than it is wide, and is situated with a solid wall to it's immediate left, taking off and immediately turning to the right will result in the rear end of the vehicle clipping the wall. The vehicle will not be stuck, but this situation does looks a bit goofy. A real vehicle doesn't work like this, of course; this problem is due to the center of rotation being at the geometric center of the vehicle, which is where it really has to be for collision detection purposes. We're hopeful that this will improve with future coding changes. In the meantime, building the vehicle with rounded corners and possibly adding solid (window contents) clip brushes to the sides of the vehicle may help appearances.
- The vehicle's sounds emanate from the vehicle's origin.
- Func_vehicles are very susceptible to being destroyed by closing doors or other moving bmodels. This will likely be improved upon, but for now... try not to allow func_vehicles near doors.
- While func_vehicle can function as a movewith parent, adding wheels makes no real sense since there's no way to stop the wheels from turning when the vehicle is stopped, and besides, this would cause the axes of the rotating wheels to be rotated, which will yield less than smooth results. It's probably best to construct a vehicle so that its wheels are hidden from the player when he's on board so he can't see that they aren't turning, or else construct a hover-type vehicle which doesn't use wheels.

[Lazarus Main Page](#)

Func_wall

The **Lazarus** func_wall is identical to the standard Quake2 func_wall brush model, with the addition of **movewith** support, which allows it to move with its parent entity, and the **count** key, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

When non-zero, specifies the number of times the wall will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

movewith

Targetname of the parent entity the wall is to **movewith**.

targetname

Name of the specific func_wall. When triggered, it will toggle on/off.

Spawnflags

TRIGGER_SPAWN Spawnflag (=1)

Wall must be called by another entity before it appears in the map.

TOGGLE Spawnflag (=2)

Wall can be toggled on and off.

START_ON Spawnflag (=4)

Sets wall to be active when the map loads. **TOGGLE** must be set also.

ANIMATED Spawnflag (=8)

Wall will display a sequence of animated textures at the rate of 4 animations/second if an animated texture is applied to it.

ANIMATED_FAST Spawnflag (=16)

Wall will display a sequence of animated textures at the rate of 10 animations/second if an animated texture is applied to it.

NOT_IN_EASY Spawnflag (=256)

Wall will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Wall will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Wall will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Wall will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Func_water

The **Lazarus** func_water is identical to the standard Quake2 brush model, with the addition of the **MUD** spawnflag, which gives the func_water (and also the **func_bobbingwater**) brush the properties of mud. This is like water, but different. When the player is in mud, his movement is restricted, and he cannot swim out - he can only walk out. Deep mud means unavoidable sinking. If submerged, the player's view is masked with a dark black-brown view. Drowning in deep mud is a distinct possibility. Movement in mud uses its own set of player sounds.

As outlined in the **redistribution** doc, if the **MUD** spawnflag is set, this entity will require the sound files:

- sound/mud/mud_in2.wav
- sound/mud/mud_out1.wav
- sound/mud/mud_un1.wav
- sound/mud/wade_mud1.wav
- sound/mud/wade_mud2.wav

The mud player sound files distributed with Lazarus are courtesy of Maulok (Anthony Bouvette).

Naturally, as with all other moving brush models, transparent textures will not move with the brush when applied to func_water. So be sure the transparent texture flag is **not** set.

Legend:
Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Direction the func_water will move on the XY plane. Default=0.

angles

Direction the func_water will move in 3 dimensions, specified by pitch and yaw. Roll is ignored. Syntax is **pitch yaw 0**. Default=0 0 0.

lip

When activated, the func_water will move [brush thickness]+[-2]-[lip value] units. Default=0.

speed

Speed in units/second that the func_water moves. Default=25.

sounds

Specifies sounds to be played while the func_water is moving. Choices are:

- 0**: silent.
- 1**: water (default).
- 2**: lava.

targetname

Name of the specific func_water.

team

Team name of the specific func_water. Func_waters with a identical team names will move together, and if solid, will all stop if one is blocked. A func_water may be teamed with a func_bobbingwater.

wait

Time in seconds for the func_water to wait before it returns to its "closed" position. If wait=-1, the

func_water must be retriggered in order for it to close. Default=-1.

Spawnflags

START_OPEN Spawnflag (=1)

The func_water will appear in its "open" position rather than its "closed" position.

MUD Spawnflag (=2)

The func_water brush will have the properties of mud.

NOT_IN_EASY Spawnflag (=256)

The func_water will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_water will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_water will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_water will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Gib fade

This isn't an earth-shattering change, but we thought we should bring it to your attention just so you'd be sure to notice it. Ever get annoyed with the game making gibs suddenly go "POOF" after 10 seconds or so? Well even if you didn't.... Next time you run a map using **Lazarus**, be sure to stick around and watch the gibs fade away. Total gib duration isn't changed, but during the last 2 seconds they'll become progressively more transparent before drifting off to Strogg heaven.

[Lazarus Main Page](#)

Hint_path

The **hint_path** is a point entity node that monsters can use as an alternative to collision detection to find their way around a level. This entity is a slightly modified version of the entity used in the Rogue mission pack.

Did you ever notice how the monsters in the Rogue MP just seemed to be a lot smarter than normal Q2 monsters? Here's a big reason why. A series of these entities, if properly placed, can assist monsters in finding an enemy once visual contact has been lost. This can make monsters appear to be much smarter than they actually are. Hint_paths are particularly useful for helping monsters negotiate their way around vertical obstacles.

So how is this thing useful? Here's an example:



The hint_paths are represented by red cubes in the picture, and of course aren't visible in the game. Assuming that the enforcer in the picture has seen the player and become angry at him, the enforcer will **never** find his way to the player in standard Quake2, unless he just gets plain lucky. Instead he'll pace back and forth on the platform he's currently standing on. If another monster on the ground floor sees the player, and the player then climbs the ladder in the corner, that monster would endlessly run underneath the player in a vain attempt to get at him. In both of these cases, if hint_paths are used, the monsters will abandon the normal Quake2 AI and follow the hint_path sequence to the player.

Here's how they work:

The **Lazarus** version of the hint_path causes a monster that has not sighted its enemy for 2 seconds to look for any hint_paths that exist within 512 units of his position. Hint_paths are set in a chain sequence, so once locating a hint_path, the monster will then examine all hint_paths in its chain. If one of those hint_paths is within 512 units of the player (or other enemy), the monster will consider this a viable hint_path chain to follow. He will move toward the closest hint path in the sequence chain and follow the chain toward the hint_path closest to the enemy. Once the enemy is sighted, he'll abandon following the hint_path chain and move toward the enemy directly.

If a monster's enemy is not visible, and no hint_path is within 512 units of the monster, or no hint_path is within 512 units of his enemy, normal Quake2 AI will be used.

If the monster has abandoned his hint_path search because he has sighted the enemy, and subsequently loses sight of the enemy again, he will not immediately return to examining hint_path chains. 5 seconds must pass between hint_path search events, to avoid goofy looping behavior that would be caused by a player ducking in and out of view.

Now, you may be saying, all this sounds really great, but how can I tell if the hint_paths as I've laid them are really working in my map? Well, we've got that **covered** too.

Hint_paths are not for every map and not for every area. Monsters may try to get to hint_paths they can see without regard if they can actually get to them or not - this is especially true of vertically-oriented areas. A monster on a ledge will consider a hint_path on the floor far below him to be mighty close, even though he hasn't a prayer of reaching it. Limit hint_path usage to areas where they can help monsters, not confuse them.

Note: The monster_medic uses hint_paths differently than other monsters; rather than use them to find the player, he uses them to find corpses to resurrect. More details on this **here**.

Unlike **path_corners**, hint_paths are bi-directional, because the chain data is built at map startup. So, it isn't necessary to duplicate hint_paths. A single chain will let a monster get from point A to point B and from point B back to point A.

Key/value pairs

target

Targetname of the next hint_path in a sequence. All hint_paths other than the last in a sequence should have a target.

targetname

Name of the specific hint_path. All hint_paths other than the first in a sequence should have a targetname.

wait

Specifies the time in seconds that a monster should pause at a hint_path before proceeding to the next. Default=0.

Spawnflags

END Spawnflag (=1)

Sets a hint_path to be a chain end. The two hint_paths at either end of the sequence chain should use this flag. END hint_paths should have a target or targetname, but not both.

NOT_IN_EASY Spawnflag (=256)

Hint_path will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Hint_path will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Hint_path will be inhibited and not appear when skill=2 or greater.

Homing Rockets

Target_blaster, **turret_breach**, and rocket-firing **monsters** and **actors** can optionally be made to fire rockets that home in on a target, adjusting their path in flight. If you play around with these rockets (see **blaster.map** and **turret.map** in the examples) you'll likely notice that slower rockets (say 400 units/sec) tend to be deadlier than rockets that fly at the default speed (650 for the player, 500 for monsters), simply because they have more time to adjust their direction.

If the rocket's target is a player, the correction factor applied to it's path is skill level dependent. On easy, you'll probably be able to jump over rockets and get away with it. On nightmare... forget about it, since the rocket will likely be able to follow your jump.

We think if you use homing rockets against monsters in your map it had better be a hellacious (technical term) battle, or it will be just plain unfair. Monsters don't know about or understand homing rockets, and they will be your basic sitting ducks.

If the player is targeted by a homing rocket, he will hear a custom "lock-on" sound. As outlined in the **redistribution** doc, if homing rockets that target the player are used, the following sound file will be required:

- [sound/weapons/homing/lockon.wav](#)

Homing rockets are damageable, so all is not lost if you find yourself in an open area and hear the "lock-on" sound. You'll most likely find that they are a bit difficult to hit. Unfortunately this is not a problem with your aiming; it is that the game doesn't keep track of the rocket position frequently enough, and as far as the game is concerned the rocket isn't really where you see it. You'll have the best luck hitting rockets with one of the bullet/shot weapons. Having said this... as far as we're concerned allowing monsters in open areas to use homing rockets against the player will most likely guarantee that the player will become frustrated with your map. Remember that it's supposed to be fun :-)

Lazarus Change Log



September ??, 2001
Version 2.1

- New Features -

Func_reflect allows you to create the illusion of mirrored surfaces in the floor, ceiling, or walls of a room. Hats off to **psychospaz** for making his floor reflection code (upon which this is based) public.

Working **func_conveyor**. Moves players, **func_pushables**, and all point entities that use a model and are affected by gravity. See the **conveyor** example map.

Lazarus **footstep** code can now use an external file to determine which textures result in what footstep sounds. So new footstep sounds may be used in **any** map, and WorldCraft users won't have to jump through hoops to add new surface flags to their maps. Also added force field footstep sounds and SURF_FORCE surface flag. Download the **footstep** sounds and the **steps** example map.

New **model_turret** is identical to **turret_breach**, but uses an .md2 model rather than a brush model. See security camera and gun turret examples in the **turret2** map.

Misc_actors now use a proper muzzle flash for shotgun, super shotgun, machinegun, and chaingun. Also, new muzzle offsets are hardwired into the code for id player models using machinegun, chaingun, or hyperblaster (standard offsets were off a bit, which you don't notice in the normal game because DM players don't use a proper muzzle flash).

Monsters can now have their **item** set to item_health_small, item_health, item_health_large, or item_health_mega.

MOVETYPE_TOSS and MOVETYPE_BOUNCE entities (pickups items, debris, gibs, grenades) will now usually bounce off of a steep incline rather than coming to rest. Although this change was made to help prevent pileups with conveyors, it should also prevent some of the silliness you sometimes see with gibs and debris.

Changed **SHOOTABLE** pickup items such that they are now clipped against monsters, actors, and insanes (in addition to players). This change was made primarily to help prevent traffic jams and embedded entities on the new working...

- Other Changes -

Pickup items that normally "droptofloor" are now dropped to the correct location instead of being embedded in the floor by 1 unit as they are in standard Q2 and all previous versions of Lazarus. This error became apparent when working on floor reflections with **func_reflect**.

Geek stuff: "noclip" now toggles player's solid state as well as movetype. So if player is MOVETYPE_NOCLIP, he's also SOLID_NOT rather than SOLID_BBOX. This is mostly useful for cheating while testing stuff. In standard Q2 a noclip player will still "touch" many moving entities. In this version of Lazarus... he won't.

- Squashed Bugs -

Fixed problem with killtargeted dead monsters and monster count. Lazarus only includes monsters that are physically present in a map in the monster count. Spawned monsters aren't counted until they actually spawn into the game, and killtargeted monsters are removed from the count... trouble is Lazarus also removed **dead** killtargeted monsters, which had the curious effect of showing total killed monsters greater than the total monster count. Oops.

Fixed a problem that prevented custom **footstep** sounds from playing when the player was walking on a slope. Thanks to james pants for pointing out the problem.

August 20, 2001
Version 2.0

- New Features -

Monster **AI changes** - monsters will run away from grenades and attempt to move out of the path of flying rockets.

New BOUNCE, FIRE_ONCE, and START_FADE spawnflags for **target_precipitation** and **target_fountain** can be used to modify the behavior of those entities.

Misc_actor can now be used as a remote-controlled robot when targeted by a **func_monitor**. Move forward/back, jump, crouch, and fire weapons, all from the safety of a remote viewing station. You also have the choice of viewing the world from the eyes of the robot or (not quite as realistic but perhaps easier to control) from a third-person perspective.

Custom debris models from *venomus* are now used by destroyed **func_door**, **func_door_rotating**, **func_explosive**, and **func_pushable**. Choose from metal, glass, barrel, crate, rock, crystal, mechanical, wood, and tech models. The crate model provided has 5 skins. All models have skin references for up to 8 skins, so if you want to create your own custom skins for debris models... well go right ahead :-)

Player-fired homing missiles. This actually uses a completely new weapon, but there's a bit of subterfuge going on in the code to give the appearance of using the standard rocket launcher with new ammo. "use Rocket Launcher" (normally bound to the 7 key), toggles between weapon_rocketlauncher and weapon_hml, assuming the player has **ammo_homing_missiles** in his inventory. As with other custom Lazarus models, ammo_homing_missiles cannot be obtained via give cheat unless *developer* is non-zero. So if you don't want players to have homing rockets, well then don't put them in your map :-). For more info see the **ammo_homing_missiles** page.

New **item_freeze** allows the owner to, in effect, stop time for up to 30 seconds. Monsters are frozen, projectiles stop in mid-air, buttons and doors cannot be opened, etc. Thanks to *venomus* for the swell pickup model.

Several changes to **misc_actor**:

- You may now "+use" **GOOD_GUY** actors much the same as you do in Half-Life. Looking at an actor and pressing +use toggles his follow-the-leader state.
- In previous versions, GOOD_GUY actors would come to the defense of the player only if there was a line-of-sight between actor and player. This requirement has been relaxed such that actors in the player's PVS will now come to the player's defense, so actors may run around a corner or 2 to help out.

- The previous +use = "get out of the way" command is now accomplished with the **go** console command. For best results you'll want to bind a key to this command, and of course explain this in your map's text file.
- Actors don't mind wading in water (but will not swim). Actor will not hesitate to enter water up to his eyeballs (his viewheight). Note that this may result in submerged actor models for short player models, but is generally the correct distance for the id models and most others.

Added ALERTSOUNDS flag to worldspawn **effects** flag. If set, monsters are alerted to player footstep sounds.

Added CORPSEFADE flag to worldspawn **effects** flag. If set, monster, actor, and misc_insane corpses begin to fade and sink into the floor after 20 seconds. You can test this flag in **any** map by setting the **corpse_fade** cvar to 1. The delay before the fade may be controlled with the **corpse_fadetime** cvar. If CORPSEFADE is set, monster_medics are removed from the game at startup.

Added JUMPKICK flag to worldspawn **effects** flag. If set, player can hurt monsters and actors and kick out func_explosive windows, for example, by jumping into them.

Added GOODGUY spawnflag to **turret_breach**. If set, turret_breach will track and fire at monsters. To prevent the turret from cleaning out a map area at startup, GOODGUY turrets are disabled at startup and must be triggered to begin tracking monsters.

Several changes to **func_pendulum**: Improved impact physics, new health and noise values.

Added health key to **info_player_start**. If non-zero, specifies the player's health when he spawns into the game at this location. This is a maximum value (player's health will never **increase** because of it). Why would you want to do this? Say you've just survived (barely) a plane crash at the start of a map...

Added **target_skill**, which allows you to build hub maps in which the player has a choice of skill level to play.

Added EXPLOSIONS_ONLY spawnflag to **func_explosive**, which, if set, makes func_explosive invulnerable to projectiles.

- Other Changes -

Removed the feature of adding player velocity to initial velocity of grenades. While technically correct, this never felt right.

Icon for **flashlight** is now displayed properly when the flashlight is picked up.

- Squashed Bugs -

Several versions ago a NOGIB spawnflag was added to **trigger_hurt**. This feature works by limiting the damage a player takes. However, a problem occurred on Easy skill: damage to the player is cut in half in the generic damage routine, so that NOGIB trigger_hurt would cause the game to creep up on killing the player rather than whacking him in one frame. NOGIB trigger_hurt now takes skill level into account before determining damage.

Func_door with non-zero accel/decel values did not act properly as **movewith** parents. Thanks to *Face the Slayer* for the heads up.

GOODGUY **misc_actors** that were defending the player would become mad at TRIGGER_SPAWN monsters who had not spawned into the game yet, resulting in the actor running in place and looking a bit foolish.

Misc_actor aiming was fouled up with supershotgun, unless the target was at the same elevation as the actor.

May 12, 2001
Version 1.9

- New Features -

Surface-specific **footstep** sounds are now possible. Shallow water, wading through deeper water, and ladder climbing sounds will be used if the worldspawn **effects** STEPSOUNDS flag is set. Other footstep sounds are played when running across brush surfaces with new surface flags: metal, grate, tile, dirt, grass, snow, carpet, and ventilation duct sounds are provided. See the steps.bsp example map for usage.

Target_precipitation allows you to add falling rain, snow, leaves, or a user-defined custom model to your map. **NOTE:** We're happy with this entity, but if abused it can very easily create overflow errors. You cannot create an intense rainstorm that covers more than a very small area. Attempt to do so at your own peril :-). The effects used in the *rain* example map are generally safe (used one at a time). You'll need to thoroughly test your maps under adverse conditions before releasing to ensure your map doesn't fall victim to *SZ_GetSpace: Overflow*.

Please note: If you previously downloaded version 1.8.3, we've made a couple of changes to target_precipitation that you'll need to be aware of. "Count" now specifies the number of models generated per second for all types, rather than models per frame. Also the "mass" value is now used only for user-defined styles; "mass2" specifies the number of splash particles. Sorry for any inconvenience this causes, but these changes are for the better.

If you've taken a look at target_precipitation from version 1.8.3, we invite you to look again for a new feature - the user-defined style opens up a lot of possibilities; all you need is the right model. Want a meteor shower that gives headaches to players and monsters? Well now you can have it.

Target_fountain is functionally very nearly identical to a user-defined style for target_precipitation. The principal difference is that the models all emanate from a point source rather than being distributed across an area.

Point_combat has a new DRIVE_TRAIN spawnflag that causes a monster touching the point_combat to drive the **func_tracktrain** at that location. He'll run you down if he can, and reverse course once he loses sight of you. There are also a few more tracktrain changes that facilitate scripted "lose control of the train and fall in a hole" sequences ala *On A Rail*. Both of these new features are demonstrated in the updated track1 example map.

Another Jed suggestion: **misc_deadsoldier** may now use any of the standard male player model skins (although without blood).

From Face the Slayer: **func_door** and **func_door_rotating** may now be destroyed. Giving doors a negative health value tells the game that the door will absorb the absolute value of "health" damage points before being destroyed. **Func_areaportal** triggering is taken care of automatically, naturally.

From blender81, monsters now sport new "gib_type" and "blood_type" values. If gib_type is non-zero, the code replaces "gibs" in gib model name with "gib1", "gib2", etc. Models are **NOT** supplied with Lazarus.

Added "effects" value to **worldspawn** to control several global changes that just don't seem to fit anywhere else.

Target_rocks has been modified to accept new models (you supply the models) by way of its style value.

A couple of changes have been made to **target_playback** to enhance our 3D sound capabilities. First, a new SAMPLE spawnflag has been added that will cause the game to use FMOD's sample functions rather than stream functions. This spawnflag is only meaningful for .MP3 files, since all other file types automatically use the sample functions for 3D sound. So what does this mean? FMOD requires the use of sample rather than stream functions in order to make use of the sound attenuation features (**target_playback distance** value). So if you have a moderate sized (**not** a multi-megabyte soundtrack) .MP3 file, you can now use all of FMOD's 3D features with it. One of the thunder sounds used in the rain example map makes use of this feature. Secondly, all SAMPLE sounds are now precached at startup. So you'll no longer have an annoying burp in the game as a sound is loaded the first time.

Added **ANIM_ONCE** spawnflag to **model_spawn**. If set, **model_spawn** will cycle through its animation sequence once, then toggle itself off. The animation sequence is repeated each time the **model_spawn** is triggered. What's this good for? For an example, see the lightning in the *rain* example map.

Freeze developer console command allows you to freeze all entities other than the player in place. This is the ancestor of what will become a new powerup in subsequent versions, and is also very useful for screenshots. Thanks **venomus** for the suggestion.

- Squashed Bugs -

Version 1.8 introduced a bug with **func_door movewith** children. The child would only move to its new location after the door had completed its move. Thanks SoftShoulder.

Ricebug correctly pointed out that the **movewith** feature didn't work properly with teleporting trains. Fixed it.

Jed pointed out a problem with **func_tracktrain** that would occasionally cause the train to flipflop 180 degrees. This should no longer be possible.

A couple of problems with **target_playback** are fixed:

- In version 1.8, a 3D **target_playback** that "movewith"ed a moving entity but was **not** currently playing would foul up the volume/location of any other 3D **target_playbacks** in the map. Of course this wasn't a problem if the map only contained a single 3D **target_playback**.
- Version 1.8 wanted to play **target_playback** sounds so badly that it would unceremoniously crash the game if **fmod.dll** was not present.

April 12, 2001

Version 1.8

- New Features -

Choo-choo! Half-Life-like **func_tracktrain**, **func_trackchange**, and **path_track**.

New spline pathing for **func_train** and **model_train**.

Added 3D, MUSIC, TOGGLE, and START_ON spawnflags to **target_playback**. Of these, 3D is the most significant, but you'll also want to check out MUSIC. What this does is allow you to distinguish between background music that the player may not care to listen to and more important voiceovers or other sounds. Also added fadein and fadeout values, which allow you to increase/decrease the volume of a sound over time. This version also supports music files (.MOD, .S3M, .XM).

Related to 3D **target_playbacks**, added "attenuation" and "shift" values to **worldspawn**. "Attenuation" controls the rate at which the volume of a sound

decreases with increasing distance, and "shift" controls the Doppler shift effect.

Func_pendulum is just what it sounds like. Check out the pendulum map in the [examples](#).

Added custom sounds to **item_jetpack**, provided by Jeff Hayat (*jeffrey*). Check 'em out in the rottrain example map. Jetpack is now selectable from the inventory.

Func_explosive now has a delay value that can be used to trigger a chain reaction of explosives without annoying SZ_GetSpace: Overflow errors. Thanks Ricebug.

Target_anger has been changed a bit so that it will no longer automatically pick the first match on its killtarget value. Instead it will select the entity that's closest to the angry dude and that has health > 0. So if you're looking to build a scripted sequence in which one or several monsters concentrates on killing an army of noncombatants (misc_insane, for example), well... you can do that now.

The code now takes fog density into account when figuring visibility. If visibility is less than 5%, a monster can no longer see his enemy. If visibility is less than 12%, the monster's aiming accuracy is decreased.

"Distance" value added to all monsters. An enemy farther away than "distance" is effectively invisible to a monster. In standard Q2 this distance is fixed at 1000 units; we've made the default a slightly meaner 1280 units. Now if you want that gladiator to rail you when you're 2000 units away instead of standing around like a slacker, well he will.

We've listened to feedback from players who aren't big fans of our perfect-aim Gunner. The first cut at reducing gunner grenade accuracy was to add a random component to his aim on easy and normal skills. However, since the guy fires 4 grenades in rapid succession the end result was to actually make him tougher, depending on how you normally play. In this version the gunner keeps the same perfect grenade aiming on all skill levels, but will only fire the 1st and 3rd grenades on easy skill, and skips the 4th grenade on normal. Your opinions are welcome.

Model_train can now be animated in the same way that **model_spawn** is.

A couple of new **developer** console commands: "spawn *classname*" adds any point entity to a map, facing the same direction as the player and 64 units in front of the player. Look slightly up to prevent embedding spawned monsters in the floor. The entity can then be moved around using the normal Lazarus **item movement commands**, and position info can be retrieved using the **id** command. Of course in the case of monsters you'll want to do this with *notarget* set. Also added "spawnself", which drops a copy of the player model in the map. This might be useful for screenshots.

- Squashed Bugs -

Fixed a problem with NO_GIB monsters. The intent all along has been that if dead NO_GIB monsters blocked the path of a moving brush model (door, train, etc.) then those monsters would explode. The implementation was off, however. This version works.

In previous versions an integer overflow problem existed with the jetpack that would cause you to be propelled very quickly to the vertical extents of the map if you had been playing the map for a while (but not using jetpack).

Previous version contained a game-crashing bug on Windows 2000 that occurred if you saved and reloaded a game while using **third-person perspective**. Thanks Nightmare66.

February 26, 2001

Version 1.7.1

Restored solidstate key for **model_spawn** to its original meaning. Solidstate=1 is nonsolid, not affected by gravity. New solidstates 3=solid, not affected by gravity and 4=nonsolid, affected by gravity.

Modified player physics just a bit from 1.7. In this version, prediction is not turned off for a player riding a brush model unless that brush model has "turn_rider" set. In version 1.7 prediction was needlessly turned off when riding **any** brush model, which resulted in a bit of a laggy feel to the game.

February 23, 2001

Version 1.7

- New Features -

Added LOOK_TARGET spawnflag to **trigger_look**. This form of trigger_look requires the player to be both inside the field of the trigger, and to be looking at the targeted entity, before that entity will be triggered. Multiple entities with the same targetname may be targeted by the same trigger_look, and only the one that's being looked at will be activated.

Added **target_playback**, which makes use of FMOD by **FireLight Multimedia** for playback of alternative audio formats, including .wav, .mid, and .mp3. A **BIG THANKS** to Firelight Multimedia for producing this product to start with, and then allowing us to redistribute their dll. This is cool.

Added "sound_restart" console command as an alternative form of the "snd_restart" console command, to avoid sound oddities when restarting the sound system when FMOD.DLL is loaded.

Added "sv_maxgibs" cvar (default=20), which specifies the maximum number of gibs that may be spawned in any 0.1 second frame. This feature will help eliminate many occurrences of SZ_GetSpace Overflow game pauses.

Added "hint_test" console command. With developer=1, look at any monster and type "hint_test" at the console. The monster will ignore normal AI and seek a hint_path, then continuously run back and forth along the corresponding hint_path chain. To return the poor guy's brain, look at him again and type "hint_test" at the console once more.

Changed "hud" console command to optionally take a state argument. "hud" toggles the hud as before, "hud 0" turns the hud off, "hud 1" turns the hud on.

Added "texture" console command to report texture name, surface flags, and value of the brush viewed. Developer must be set to 1 to use.

Added "style" key for info_player_start, info_player_deathmatch, and info_player_coop. If non-zero, this value specifies the player's starting weapon in the same way the worldspawn style key does, and overrides the worldspawn setting, if any. For map transitions, a non-zero style value for info_player_start will remove all weapons and ammo from the player and only give the weapon (and ammo for that weapon) specified by style.

In most cases the player should now be able to ride pitching and/or rolling brush models without being eaten by the model. This problem has always been with Q2 and is likely a big reason you don't see many rideable rotating entities. For flat platforms the code is fairly foolproof - you should **never** take damage from the brush you're standing on. There is, however, still a problem with vertical components of brush models. Lazarus currently pushes a player up until his bounding box won't intersect any of the model's brushes... which beats being eaten by the model but may result in the player being tossed off of the model, particularly if it is translating

as well as rotating.

Added REVOLVING spawnflag (=2) to `func_door_swinging`. If set, the door is always "closed" and will rotate in the same direction when opened or used again. However, the direction of rotation is now determined according to which side of the origin the activator (or damage for shootable doors) is on, rather than simply being tied to which side of the door the activator is on. This feature will come in handy when used in conjunction with "the next big thing". Stay tuned.

`Func_pushable` with `health<0` option makes it truly indestructible, and will block moving brush models. It will likely take a bit of trial and error to get the brush model speed adjusted such that it will appear to touch the `func_pushable` when blocked. The trick is to set the speed such that (travel distance minus blocker dimension + 2) divided by the speed is exactly equal to or slightly less than an integer multiple of 0.1 seconds. For example, a 128 unit tall door with `lip=8` that moves vertically will come to rest on a 64 unit tall `func_pushable` with `speed=97` (0.598 seconds) or 116 (0.5 seconds).

Improved `hint_path` code; it's now much more effective. The original code was designed to find the `hint_path` closest to a monster's enemy as the monster's goal, but... didn't. This didn't cause any problems, but **did** cause monster to restart the `hint_path` sequencing code more often than was necessary.

Several changes to medic AI (these guys are really annoying now):

- The most significant change is that when leaving his idle state (**only** after being awakened by being triggered or angered in some other way) the medic will now look for a `hint_path` entity, and if found, follow the corresponding `hint_path` chain looking for dead monsters to resurrect. To facilitate this a bit, if medic has not seen or heard his enemy in more than 20 seconds he'll return to his idle state, which means he'll potentially give up on finding the player and go follow `hint_paths` instead. Just as angered monsters never give up looking for the player, such a medic will never give up looking for buddies to resurrect.
- Medic recognizes transparent brushes (like windows) are solid, and will not dumbly attempt to heal a monster on the other side.
- If medic cannot reach his intended patient in 5 seconds (e.g. if the patient is on the opposite side of a lava pool), he'll now look for a `hint_path` that leads to him. If found he follows it.
- Medic will now give up attempting to heal a monster after 10 seconds (if, for example, the monster is more than 400 units away and the medic cannot find a path to the monster). Medic will not attempt to heal that monster again for at least 60 seconds.
- Added a check to ensure that a clear line-of-sight exists between medic "muzzle" and the dead monster before initiating cable attack. This is **not** a guarantee of success, since medic animations cause the cable origin to move around a bit, but it should prevent a few occurrences of medic attempting to heal a monster when the cable can't get there.
- Added "`medic_test`" console command. When turned on, medic will begin moving to `hint_paths` **without** waiting to be angered. This feature is of course best used in conjunction with "`notarget`" to see the medic's fancy new moves.
- Healed monsters now display the red shell for the length of the medic's cable attack animation, which seems to be the intent of the original code.

Added ability to use custom skins (along with the original skins) for all `monsters`. The skins are up to you, though... artists we definitely are not. This feature makes use of Argh!'s player model transmogifier code.

`Target_bmodel_spawner` is a new point entity that clones existing brush models. The intention here is to help prevent ERROR:Index overflow.

Added CHASE_CAM spawnflag (=1) to **target_monitor**. If set, camera will follow the target_monitor's target by "distance" units, raised "height" units above target's origin. Default distance=128 units. Note that "movewith" is incompatible with CHASE_CAM (if both are used, CHASE_CAM is turned off).

TRACK Turrets are **much** deadlier now. In previous versions turret would set its angles based on the target in the **previous** (rather than the **current**) frame. Turret AI is tied to skill level. Skill 0 is the same as standard Q2: turret finds desired angles but does not apply the corresponding changes to angular velocity until the following frame. On skill 1 those changes in angular velocity are immediate. Skill 2 and 3 are the same as skill 1, except that the "target" is a prediction of the target's position 0.1 seconds in the future.

For **target_laser**, "mass" value now specifies the laser diameter and overrides the FAT spawnflag. Mass must be greater than 1. If mass=0 or is left blank, the default diameters will be used (4 for normal, 16 for FAT). "Style" value may be used to 1) create a no damage laser (style=1, 2, or 3), 2) create a laser that monsters will ignore (style=2 or 3), or 3) create a laser that doesn't produce sparks at the receiving end (style=3).

Added ability to use a **target_anger** to make monsters attack a brush model. Brush models used for this purpose **must** have an origin brush.

Added ENVIRONMENT spawnflag (=64) to **trigger_hurt**. If set, players using the environment suit will not be damaged by this trigger_hurt.

Changed GOOD_GUY behavior slightly in that if a player triggers a GOOD_GUY, that flag is turned off and the monster attacks the player as a normal monster would. This change avoids a bit of silliness in that used GOOD_GUY monsters will no longer attack the player, only to then come to the player's defense if the player is attacked by someone else.

- Squashed Bugs -

Fixed a horrible bug that has existed in the last several public releases. The problem was associated with saving games. As part of an optimization frenzy several versions ago, saving a game caused "player_noise" entities to be removed. This was done to prevent duplicate player_noises from spawning into a map that you had left and returned to. The change worked fine for the intended purpose, but was A VERY BAD THING when saving a game in the middle of a map and continuing to play. This bug may have caused newly spawned entities to appear at the player's location when he jumped or fired his weapon, or may have caused unexplained crashes when alerting unseen monsters.

Solved a problem (we hope) with switching from software or "Default OpenGL" to "3dfx" in maps with fog. The problem with changing video while fog is active is that the gl_driver setting is updated before the actual vid_restart, so that the code would send Glide instructions to your card before it was ready. This resulted in a hardware crash with Voodoo 3/4/5 requiring a cold restart and a slightly less obnoxious game crash with a Voodoo 2. With this version, the code preserves the frame number when using software rendering or when OpenGL or Glide fog is used, and will NOT use 3dfx fog calls until at least 1 second has passed since the last time the code detected software or OpenGL. When switching to 3dfx you may notice a short hiccup with no fog, followed by the correct fog. Why does this matter? Well if you switch between rendering dll's with a Voodoo, in order to use gl_showtris, and the map has fog, it won't go boom no more.

Fixed a problem with collision detection for brush models that do **not** use an origin brush. In previous versions, in some circumstances a func_pushable would not block the path of a closing door or moving train when it should have.

In previous versions if a trigger_fog was active when a game was saved, and that

game was later loaded, the fog parameters were a bit unpredictable.

Fixed a fix. Several versions ago we fixed a bug in standard Q2 that resulted in rotating bmodels that were nowhere near dead monsters gibbing those dead monsters when the player would walk through them. Unfortunately this fix resulted in ALL moving brush models now simply passing through dead monsters. Fixed it, happily w/o breaking previous fix.

A few long-standing problems with medics are fixed.

- The first problem has been around since the original game. When medic heals a monster, the healed monster's enemy is set to the medic's oldenemy, if oldenemy is a player. If medic's oldenemy is NOT a player (for example it's another dead monster), then the recently healed monster is completely oblivious. Get in his face and he just smiles until you jump or fire. This problem was due to the monster preserving his enemy - he still was mad at you, he just didn't know it :-). So now... if medic doesn't set a healed monster's enemy, the enemy is set to NULL and the monster will find an enemy through normal AI stuff fairly quickly.
- Another original game bug was the occasional persistence of the red shell on revived monsters. Up to now the code has depended on the medic, not the healed monster, to turn the shell effect off. Well it looks to me like it should work, but still sometimes does not. Short version is the code no longer depends on medic to turn the effect off, but on the monster to do so.
- A Lazarus problem with the medic which arose from making one change without beefing up something else is now also fixed: Lazarus medic prioritizes targets differently than the standard game - if a dead monster is within 1000 units, he'll always attempt to revive that monster rather than attack the player. This has been in place for a couple of versions. Problem was if the medic was within 1000 units of dead monster but farther away than the max healing range (400 units) he'd take a couple of baby steps toward the dead monster every 3 or 4 seconds then stop. This was particularly noticeable if you used "notarget". In this version under the same circumstances medic will run to within the max healing range of the dead monster.
- Also, code now checks for other monsters or the player being entangled with a dead monster. If another entity intersects a dead monster's bounding box, the medic will not attempt to revive that monster.

Misc_actor with a combattarget was a moron. Recent "ideal range" code made it impossible for the actor to decide between running to a point_combat and running to his enemy. Thanks Jedi for the bug report.

Fixed a minor annoyance with **turret_breach**. Depending on turret_breach and turret_base construction, in previous versions when operating a player-controlled turret and looking down, the game would play footstep sounds as though the player were running downhill.

Also for turret_breach, fixed a problem with grenade-firing turrets with low grenade velocities (distance value). Previously the grenade aiming calculations tended to get stuck in an endless loop searching for a good trajectory. Also eliminated an annoying jitter in grenade-firing turrets that would occasionally show up when the player was at a very close range.

Fixed a mistake in initial weapon selection code (**worldspawn** "style" key). Previously if style<0, no ammo was given for the default weapon.

Fixed a zoom-related problem with joy_cvars. Previously these cvars were always forced to default values at startup.

Both **trigger_fog** and **target_fog** were fouled up (but in different ways) if both a delay and a 180 deg. density were used. Thanks Argh!

Fixed a problem associated with OWNED_TURRET **trigger_look**. In previous versions player position was not updated as the turret turned. The view was correct, but the invisible non-solid "player" may or may not have been in the correct position to trigger the **trigger_look**.

January 18, 2001
Version 1.6

- New Features -

Added **func_door_swinging**, a rotating door that always opens away from its activator.

Added dmg key to **func_force_wall**. If non-zero, touching an active **func_force_wall**... hurts.

Func_vehicle driver may now look around while driving. Driver still turns with the vehicle. **Func_vehicle** can act as a **movewith** parent. Added **turn_rider** key to **func_vehicle**, which effects any **movewith** children (but not the driver).

Func_water and **func_bobbingwater** have a new MUD spawnflag. Mud slows player progress considerably and will pull the player in if deep enough.

"hud" console command toggles the display of the HUD. Useful for demos and screenshots.

Added **item_jetpack**. Fly around your map.

Misc_actor:

- New player model transmogrification code from **Argh!** now makes it possible to use id male, female, and/or cyborg player models without redistributing those models. This is very cool.
- Modified accuracy of actor firing, which previously was too sloppy.
- Added "run away and hide" code for actors with low health. This feature can be disabled by setting the **actorchicken** cvar to 0. Chicken actors find new courage if their enemy is attacked by someone else.
- Actors now seek cover immediately after completing their attack sequence. Actor will pause in his hiding spot from 0-6 seconds before resuming attack. This feature can be turned off with the **actorscram** cvar.
- Added quite a bit of smarts to actor decision-making concerning his preferred range to target. You should no longer encounter protracted battles between two face-to-face actors.
- Rocket-firing actors fire at the target's feet roughly half the time. For all other weapons actors fire at the target origin. In both cases, actor first checks to see that the desired target is visible.

Model_train, **misc_viper**, and **misc_strogg_ship** are now all capable of using the rotating abilities of **func_train**. **Misc_viper** and **misc_strogg_ship** can now be damaged by weapon fire.

Monsters:

- **Monster_soldier_X** accuracy is now a function of skill level. Easy=same as standard Q2. Nightmare mode gives perfect aim.
- Monster-fired rocket splash damage radius is now skill-level dependent, making tougher skill levels quite a bit deadlier.
- If enemy is not visible at end of **monster_chick**'s windup phase, she'll now fire at the feet of enemy's last known position.
- **Monster_soldier_light** blaster bolt speed is now skill level-dependent, = $600 + \text{skill} * 100$.
- **Monster_chick** now goes through her windup sequence a bit faster, depending

on skill level. Out of the 13 frames, 2 frames are skipped on normal, 5 frames on hard and nightmare.

Added **target_change**, which can be used to alter characteristics of other entities. For example, you can change a path_corner's target to a different path_corner, so that a func_train follows a different path.

Added NOGUN spawnflag to **target_changellevel**. If set, on the next map or demo the cvars crosshair and cl_gun will be 0. These get reset to previous values when exiting that next map or demo.

New **target_fade** allows you to fade the screen to a specified color and opacity.

Added **target_failure**, a mission-ending event used to inform the player that... he messed up. In conjunction with this, added NO_BACKGROUND spawnflag to **target_text**.

Target_movewith can be used to tie the motion of one entity to another, or remove that association.

Target_set_effect is used to apply special effects to point entities.

Target_sky allows you to change environment maps during the game.

Added Rogue's **trigger_disguise**, which effectively makes the player invisible to monsters until the player fires a weapon or another trigger_disguise with the REMOVE spawnflag is touched.

Added NOGIB spawnflag to **trigger_hurt**. Changed trigger_hurt code to check for the presence of a damageable entity at activation. In standard Q2, if a monster is standing inside a trigger_hurt when it is activated, he won't "touch" the trigger until he moves or reacts to the player.

Added SILENT spawnflag to **trigger_key**. If set, neither the success or failure sounds are played, and the "You need the ..." message is not displayed. This is useful for removing items from players.

Trigger_transition may now be used with LANDMARK **trigger_teleporter** and **misc_teleporter**.

Changed **turret_breach** such that it will rotate with a "turn_rider" movewith parent **unless** it is under the control of a player/monster. Turret_breach can now fire grenades and hyperblaster bolts.

- Squashed Bugs -

Fixed a standard Q2 bug that caused rotating brush models to gib a dead monster even though the monster was **nowhere near** the rotating brush. This would usually occur when the player would walk through the dead monster.

Changed monster jumping code so that monsters/actors will NOT attempt to jump if their path is blocked by another monster, actor, or the player. The code has always prevented monsters from intentionally jumping onto other monsters or the player, but monsters were still very prone to attempt to jump **over** crouched monsters. While entertaining, this was a bit goofy. One unexpected jump side effect that we'll likely leave in... it takes a bit of patience, and circumstances have to be just right, but if you fire rockets at fast-moving monsters like the berserker you'll see it sooner or later.

Solved a long-running feud with the Q2 physics that now allows you to ride a turn_rider rotating platform without an exceptionally jerky ride. Take a ride in the helicopter in the lcraft example map.

Fixed a game-crashing bug that occurred when shooting GOOD_GUY **misc_actors**.

Func_vehicle now works correctly if given an initial yaw angle. Also, **func_vehicle** will now plow through multiple damageable entities rather than being stopped cold by the first obstacle.

Added a workaround to make **misc_blackhole** visible. This is a strange error shared by the Rogue MP... no idea what's going on here, but hey, it works.

Target_changelevel with CLEAR_INVENTORY spawnflag now properly restores player health to 100 and forces a weapon change to the blaster (or no weapon, depending on **worldspawn** style key).

Fixed a problem with **target_spawner** due to changes associated with **trigger_transition**. Gibbed monsters had their classnames changed to "gibhead" to facilitate map transitions. This was fine, but the **target_spawner** code formerly used the same address for the spawner's target field and the spawned entity's classname. This resulted in correctly spawning a monster the first time, but if/when the monster was gibbed then subsequent **target_spawner** uses would spawn a gib head rather than a monster.

Fixed a game-crashing problem with **dmgtteam** monsters/actors. Previously if a **dmgtteam** member was alerted to come to his pal's defense but could not see the attacker, under some circumstances the game would suddenly crash to the desktop.

Fixed a problem with **model_spawn** that prevented a NO_MODEL **model_spawn** from being toggled. Thanks SoftShoulder.

Fixed another problem with **model_spawn** (hopefully permanently this time) that would cause a solid **model_spawn** to crawl up your leg when standing on top of it.

- Other Changes -

Added a feature to help track/prevent Index Overflow errors. If **readout=1**, **dll** displays model and sound indices as they are used. **Readout** MUST be set on the command line or in a **.cfg** file; once the game starts it is too late. For best results start the game with "+set logfile 2 +set readout 1". After walking through the map, go to a DOS prompt in the game folder and type "sort qconsole.log >index.txt" (or whatever name you like) to make the list a bit easier to read.

November 3, 2000

Version 1.5.1

- New Features -

Several **monsters** and **misc_actor** can now jump up or down to reach their intended target, with no intervention from the mapper. Berserkers are much happier now.

Monster AI - not-so-dopey-after-all monsters won't run into lasers or run closer than their present position to live grenades.

Added **IGNORE_FIRE** spawnflag to **monsters** and **actors**, which basically tells other monsters/actors to ignore friendly fire from this guy. This will keep monsters focused on what they do best - killing marines rather than fighting among themselves.

ANY non-flying, non-exploding **monster** may now have flies at death. Or even before, heh. We've also fixed a bug in standard Q2 mutant and infantry code, which caused those monsters to ALWAYS display flies when the original intent was to have flies only half the time.

Expanded **misc_actor** GOOD_GUY behavior to give those guys a bit more freedom in killing monsters. After knocking off available enemies, GOOD_GUYS will then follow

the player around to provide further assistance. Also added +use feature to tell GOOD_GUY actors to get out of the way.

If bounding box coordinates or muzzle offset locations for `misc_actor` are omitted, default values for `bleft`, `tright`, `muzzle`, and `muzzle2` shown in the `ActorPak` documentation will be used. This should be a nice time saver, assuming you use an ActorPak model and you agree with our defaults.

`Misc_viper_bomb` can be used multiple times and can specify the entity that "drops" it, rather than always using the first `misc_viper` in the map.

Fog. `Trigger_fog` can now start off, be toggled, and have a delay value. Both `trigger_fog` and `target_fog` can change density according to the player's viewing direction.

`Path_corner` wait < 0 now properly causes monsters/actors to wait at the `path_corner` until another external event causes them to move.

Added `target_monitor`, a viewing station that can be used to show remote events to the player.

Added `target_animation`, which can be used to force actors and other entities to play a specific animation sequence. This might be useful for scripted sequences, and can also be used to give actors a bit more personality.

Added `trigger_teleporter`, a brush model teleporter that can be any size.

`Trigger_bbox` can now be made shootable (while remaining non-solid).

`Func_door` can now act as a `movewith` parent.

Added `trigger_transition`, a trigger field that allows monsters, weapons, and other point entities to change levels along with the player due to a `target_changelevel` activation.

`Trigger_hurt` and `target_laser` heal the touching entity if `dmg < 0` without the annoying damage effects (blood, "Arghhh!!"). Also, if `dmg < 0` and `SILENT` spawnflag isn't set, `trigger_hurt` plays the `item_health_small` pickup sound rather than `electro.wav`.

- Interim release new features -

You can now specify which weapon the player spawns into the game with (or choose no weapon at all). Please note that this feature is strictly experimental at this point, and we do not recommend using it in released maps yet. Choice is made with the `worldspawn` "style" key, -1=no weapon. As of now, there's no way to force unknown player models to use the "no weapon" `w_null.md2` for a weapon model, so in DM or if using thirdperson view, unless the player is using the male model he will appear to be carrying his model's default weapon (`weapon.md2`) even though he really doesn't have any weapon at all.

Added `weapon_blaster`, which really only makes sense to include if the player starts with no weapon.

Added `LANDMARK` spawnflag (=64) to `misc_teleporter` and `trigger_teleporter`. If set, the user's angles and velocity are preserved, rotated if necessary by the destination angles minus the teleporter angles.

- Squashed bugs -

`Path_corner` now distinguishes between `path_corner` and `target_actor` for its next target, so that `target_actor` JUMP spawnflag doesn't cause a teleport.

Fixed problem with **target_anger** that prevented pathing function from working unless a killtarget was also specified.

Fixed a standard Q2 bug with **trigger_hurt** that would prevent normal sound from being played 90% of the time if the "slow" spawnflag was set.

- Interim release bug fixes -

Fixed a nasty bug affecting 3dfx users only. Previously if a 3dfx user entered the game using software rendering, the code inappropriately turned fog off using a Glide call, which... was not good.

Trigger_transition:

- Save/Load games didn't work at all, or putting a good face on it, worked too well. Entities which had changed levels were duplicated - one in the spot where the game was saved, one in the location where the map first started up.
- Monsters in the middle of a death animation at the time a target_changelevel occurred were stuck in that frame until returning to the previous map.
- After switching levels then switching back, misc_actor lost his weapon.
- Dead actors who changed levels regained a stationary weapon along with a stream of R_CullAliasModel errors.
- Code now correctly accounts for users running Lazarus from baseq2 folder when writing/reading entity changelevel file.

Misc_actors temporarily lost their minds after switching weapons, and would not fire until fired upon.

If a player acquired a new weapon immediately prior to changing levels, so that the new weapon didn't have time to come up, you'd be flooded with a stream of R_CullAliasModel errors in the new map.

In previous version if you triggered a "turn_off" target_fog with a delay before any other fogs were activated, a solid black fog faded away. In this version the same situation results in... nothing, which is correct.

Not so much a bug, but an improvement: fake player replacement for player using a target_monitor or func_monitor is now animated using the player's stand animation, rather than being frozen in place.

October 1, 2000
Version 1.4

- New Features -

Added **misc_actor**, a combatant with a choice of models and weapons that can be a good guy or a bad guy.

Added Rogue's **hint_path** entity, a device intended to assist monsters in tracking down players, thereby making monsters appear quite a bit smarter than in the standard game.

Added LANDMARK spawnflag to **target_changelevel**, allowing you to have HL-like transition zones from one map to the next.

Added **target_cd**, which allows you to specify a CD track to play a specified number of times.

Added OWNED_TURRET spawnflag to all trigger field entities (trigger_once, trigger_multiple, trigger_look, etc.) This allows a player who is currently viewing the map through a turret_breach camera to "touch" a trigger.

Target_anger now includes the scripting capabilities of **target_actor**.

Several features added to help make the game run more efficiently and prevent "ED_Alloc: No free edicts" errors:

1. **count** key/value added to many entities, allowing you to easily remove entities from a map after being used a specified number of times.
2. **Target_locator** now correctly removes itself after performing its function at startup.
3. **Trigger_key** without the Multi-use spawnflag is removed once used.
4. Single player game no longer reserves 8 slots for "bodyque" entities, which are **only** used in deathmatch and coop games.

Added FLIES spawnflag to **misc_deadsoldier**.

- Squashed Bugs -

Sitting, decapitated **misc_deadsoldiers** exploded when gibbed in previous versions. This was due to a conflict between **misc_deadsoldier** spawnflags and **NO_GIB** spawnflags for monsters.

Fixed a bug in **target_text** that resulted in a crash if a line of text consisted of 35 or more characters with no spaces. Also added support for \n line breaks and fixed a number of inconsistencies with word-wrapping and formatting codes.

Fixed a game-crashing bug with **target_anger**. Previous versions would cause a non-combatant killtarget to crash the game when "reacting" to damage.

Solved a game-crashing bug that would sometimes occur if using **third-person perspective** during a level change.

September 4, 2000
Version 1.3

- New Features -

Added **target_attractor**, a sort of entity magnet with many potential uses.

Added **trigger_look**, a special-purpose trigger_multiple that activates its targets only when the player is looking at a specific area.

Added **trigger_bbox**, a point entity that allows trigger field functionality without a brush model.

Improved performance of the **movewith** feature of several entities, especially **func_door** and **func_button**. Also added support for many new entities.

Added **console commands** that allow a mapper to move entities to the desired location in a map. Mapper can then use the **id** console command to get the entity's position and other information.

Func_pushable can now be dropped on monsters or players, resulting in damage and/or death to the squashee.

Floating **func_pushable** now takes a rider's weight into account. 64x64x64 crates will generally sink about 6 units with a player rider; 32x32x32 crates will generally sink to the bottom.

Added recursive brush model-rider movement code. In other words, a player standing on a **func_pushable** box on a box on a box on a moving **func_train** will move along with the **func_train** (as do the **func_pushables**, of course).

Added custom sound spawnflag to **trigger_push**, so that you are no longer limited to

using the default wind sound.

Target_locator now copies the angles value of the path_corner to the moved entity (for point entities only, not brush models).

- Squashed Bugs -

Corrected a problem with **func_pushables** riding a rotating brush model. Note that if the brush model has **turn_rider** set, then the func_pushable **must** use an origin brush.

Solved a map order bug related to target_lasers with **movewith** func_trains.

In previous versions, monsters placed in the map with a **target_locator** lost their minds and basically became statues. Fixed it.

- Other Changes -

Func_pushable bounding box has been changed slightly. This change is for the better, but **may** require you to re-work old maps if you've used stacked func_pushables.

id and **bbox** developer console commands now find pickup items as well as solid brush models and point entities.

Decreased height of monster_gladiator bounding box by 16 units to more accurately reflect model dimensions. This was done mainly for the benefit of dropping things on the gladiator, but might also now mean that a gladiator will have access to map areas that he previously could not enter. All other monster dimensions are more or less correct, with the exception of the floaters. We've all seen floaters bury their heads in the ceiling. The only way to correct this problem is to increase the size of the bounding box, but this change might result in breaking existing maps.

July 30, 2000
Version 1.2

- New Features -

Added **movewith** key to many entities. If used, the entity's origin, velocity, and rotational speed is relative to its "movewith" func_train or model_train. Future versions will most likely allow additional entities to serve as *movewith* parents.

Added SMOOTH_MOVE spawnflag to **func_train**.

Added **turn_rider** value to all rotating brush models. If non-zero, players standing on that brush model will rotate with the model. Also added a turn_rider server cvar, so you'll be able to turn this feature off for network games (where prediction might foul things up a bit) or if you just don't like turning along with rotating objects.

Added **target_effect**, a souped-up target_temp_entity.

Added **func_force_wall**, a force field with a new particle effect from the Rogue Mission Pack.

Minor changes to target_rocks: It is now much less likely that a rock will get hung up on the side of a cliff and spin in place. Rocks now use the same 2-second fade that Lazarus gibbs use, rather than going "poof!"

Fixed a problem in original Q2 monster AI - trigger spawned monsters could not target a path_corner.

Added noise fields to **model_spawn** and func_rotating so that these entities will play a sound when active.

Player velocity matches the entity he just jumped from, so, for example, jumping up won't cause the func_train you were standing on to run out from under you.

Player velocity is added to initial grenade launcher grenade velocity. So if you're running after the helicopter in the example maps trying to lob a grenade through the door, you might actually succeed :-). We're interested in your opinion on this change, so if you have a gripe please let us know.

Changed NO_GIB **monster** damage effect from TE_SCREEN_SPARKS to TE_SPARKS.

- Squashed Bugs -

Fixed a bug in **tremor_trigger_multiple**. After toggling off, then on, the trigger reverted to normal trigger_multiple behavior.

Ack! Fixed a bug introduced when fixing a Q2 bug. Previously, a path_corner w/o a target that was targeted by a monster would crash the game, even if wait was set to -1.

July 18, 2000

- New Features -

Version now bumped to 1.1.

Added rotating **func_train**. Big thanks to Rroffstein (Martin Painter), who provided the basis for this code. See the whizbang **rottrain.bsp** in the **examples**.

Fixed several annoying bugs in the original Q2 source concerning **monster** movement 1) away from path_corners with a wait value and 2) to initial point_combats.

Added CUSTOM and LOOP spawnflags to **target_lightramp**. These changes allow you to use any pattern you want when switching lights on/off. This also effectively allows switched flashing lights, which isn't possible in standard Quake2. See **lramp.bsp** in the **examples**.

More smart monster stuff: Monsters will now run away (and possibly hide) when damaged by anything that they don't normally get mad at - closing doors, trains, lava, target_blasters, etc. This will continue to improve. An example of this new behavior is shown in **monsters.bsp** in the **examples**.

- Squashed bugs -

Fixed several problems with turret_breach (it's perfect now :-)). Under some circumstances blocking the movement of a TRACK turret_breach would cause a game crash. Also (this was really annoying) normal **turret_drivers** were animated along with the turret_breach they were operating, resulting in some very peculiar dance moves.

Fixed a problem with **fog** for normal GL cards. In previous version the fog was **extremely** dense on these cards, even though it appeared more or less right using the Mesa GL drivers on a 3dfx card.

Fixed a problem with fog being retained across a target_changelevel or gamemap command. Fog will now be turned off when switching maps with either the gamemap command or a target_changelevel (this was always true with the map command). One possible **source of confusion** here: if you stay within the same unit and leave a map that has fog on and then return to that same map, fog will be on whether the map you just left used fog or not.

Ack! If you downloaded lazarus.zip since Friday July 14, you've undoubtedly noticed that the rottrain example immediately crashed. This wasn't a code problem but a

pak-editor-ate-my-textures problem. This has been fixed.

In previous versions if the player managed to jump on top of a solid model_spawn, the model_spawn would... umm... crawl up the player's leg.

In previous version if the player exited a trigger_fog while a "turn off" target_fog with a delay was still ramping down, results were... not as expected. This works correctly now.

- Other Changes -

Player must now use +moveup (jump) to exit a player-controlled turret_breach. In previous version, +back was used. However, for users who don't normally use freelook this made the turret exceptionally difficult to control.

Player riding on a floating func_pushable is now transported with the func_pushable. More importantly, we've eliminated the annoying jitter when riding a func_pushable.

July 6, 2000

Umm... changed our name, and welcome to our new home. Concurrent with this event, version numbering is begun with 1.0.

Several minor bug fixes and one major PC-crashing bug fix to the turret_breach. Previously, if you were using a func_monitor to look through a turret_breach and the map was changed due to a target_changelevel or console gamemap command... well let's just say the result was not good.

Added followtarget key to turret_breach. If no player or monster targets are found, turret will follow (not fire at) the entity specified by this value.

Added machinegun fire to turret_breach.

Several procedural changes and one visual glitch fix to monster_medic.

A couple of transparent changes to zooming make the thing operate a bit more logically.

June 28, 2000

Changed a few things up to help prevent borking your m_pitch setting when using zoom; also removed the necessity of using dhpak.cfg.

Minor changes:

- In previous version, if you had notarget set and switched to third-person view, the fake player did **not** have notarget set and was fair game for monsters and automated turrets.
- Zoom is automatically turned off when you switch to third-person view.
- If you are zoomed in while controlling a turret_breach via a func_monitor, the turret_breach will not grab control back from you. Normally, an automated turret takes over control if the viewer does not change the view angles for 5 seconds.
- Added a fake crosshair to the zoomed view. Think you can do better than \pics\zoom.pcx? I bet you're right... how 'bout sending me a pic?
- In zoomed view, "muzzle" is moved to the player's viewpoint, rather than being placed at the normal 8 units below the view. If you place the crosshair on a target with a hitscan weapon... you **will** hit it.

June 27, 2000

As promised, with this version we have an example map illustrating some of the very cool things you can do with target_rotation. See rotation.map in the example map

files.

Still more changes to **turret_breach**:

- You can set a spawnflag on **turret_driver** to make him operate the turret_breach remotely.
- Turret_breach is now animated when "owned", either by a turret_driver, a player driver, or a player viewing the turret_breach with a **func_monitor**. What's this useful for? How about a pulsing light to indicate when the turret_breach is being activated by another player/monster (as opposed to an automated turret)?

New continuous **zoom** feature that adjusts mouse and joystick sensitivity. This is pretty slick, but only works for single-player games and requires you to use a custom cfg (distributed with DHPak).

Changed code so that trigger spawned monsters are not counted in the total monster count until they actually spawn into the game.

June 22, 2000

A few more changes to the turret_breach/func_monitor combo:

- Player must be within 100 units of the func_monitor to use it.
- Automated or remotely controlled turret breach will not switch targets unless the new target is at least 100 units closer than the old target. This change helps prevent a pair of agitated berserkers from making you lose your lunch :-)
- Automated turret (and target_blaster and target_laser) will **not** acquire an invisible player (e.g. player embedded in a camera) and they **will** acquire the fake player taking the real player's place at the func_monitor.
- Added support for a "count" value for the func_monitor's target to specify the cycle order for cameras. If the count value is 0, the map order is used. This change will help the mapper correctly fill in the "viewmessage" value for the camera, e.g. "Camera #3".

Misc_teleporter adds 3 new spawnflags: START_OFF, TOGGLE, and NO_MODEL. It may also have multiple (up to 8) targets. The code currently selects a random destination from among the 8 or fewer entities that are targeted by the misc_teleporter.

June 21, 2000

Turret/monitor goodness:

- If the player is damaged while using the func_monitor, he's automatically switched out of camera mode.
- Multiple cameras may be accessed by func_monitor. Switch between cameras with the +moveleft and +moveright keys.
- Extreme trouble but worth it - the player will now see the correct muzzle flashes and hear weapon sounds from the camera's perspective. This part was no trouble, but then making monsters attack the fake player's body at the monitor was... interesting :-)
- Added "viewmessage" value for turret_breach. This message is displayed when a player uses a func_monitor for remote viewing.

Added **target_rotation**. This entity allows you to target an entity from a selection of entities, either in a sorted order or picked randomly. What's this useful for? Stay tuned.

Also fixed a problem with third person view: weapon change caused weapon to be displayed in the camera view.

June 19, 2000

Quite a few changes to **turret_breach**. You can create an automated turret ala Rogue, remotely control a turret with the new **func_monitor**, make a nifty security camera with a new "sounds" value, and create a popup turret using the new TRIGGER_SPAWN spawnflag. The NO_DRIVER spawnflag has been eliminated.

June 17, 2000

Fixed minyaw and maxyaw values for turret_breach. You may not have noticed before, but under some conditions these values didn't work properly, either in DHPak or the original game.

Added NO_DRIVER spawnflag to turret_breach for Rogue-like automated turrets.

Homing rockets now automatically blow up after flying for 8000 units. This is the same thing normal rockets do, only the homers die with a no-damage explosion rather than simply going *poof*. This change is designed as a failsafe for you guys with the quick feet who can manage to outmaneuver these things... it helps prevent game errors caused by having too many homing rockets flying around simultaneously.

June 16, 2000

Added lockon sound for homing rockets. Only the target of the homing rocket will hear the sound.

Trigger_relay may now include a message and sounds values.

Added dmgteam value for trigger_relay. Trigger_relay will be triggered whenever a monster with a matching dmgteam is injured.

Target_speaker - if attenuation is set to -2, only the player who activated the speaker will hear the sound.

... and please stay tuned. Mad Dog has hit me with a large collection of extremely cool things to do, coming this way soon.

June 13, 2000

Bah... my expert version control system works about the same as everything else I do. Basically it goes like this: "Well let's see how **THIS** works". Yesterday's distribution did **not** include homing-rocket firing monsters, despite documented promises to the contrary. Today's version does... I think :-)

June 12, 2000

You want tougher monsters? We bring you tougher monsters. :-) This is the first in a series of changes designed to make the monsters smarter, better opponents. In this version:

Gunners are **much** better grenadiers. Mad Dog suggested changing the gunner AI so that they would not fire grenades if the target was above them (which is what the Rogue MP does). But I thought... why stop there? Why not make them smarter, and have them **figure out** whether they can lob a grenade to a specific position and, if not, use the machinegun instead. So they do that. But much more importantly, if they **can** shoot you with a grenade, they do so much more accurately than either the original Q2 game (which was done really badly) or the Rogue Mission Pack, which was quite a bit better but still not correct. DHPak figures out the correct launch angle to get a grenade to the target. See the **monsters** section for a full description of the gunner's new AI. In case you were wondering, I'm pretty pumped up about this and other changes coming soon. See the last 2 rooms in monsters.map for an example of

the new gunner AI. You'll notice that the gunners on the lower level won't fire grenades at you when you're on the top 3 levels - because grenades can't make it there. But step on down a bit and... heh... see for yourself. God mode is highly recommended :-)

Rogue source, with a few changes: Tanks and chicks do not fire suicidal rocket blasts into adjacent walls (which might happen with standard Q2 if the player strafes out of view). They will lead the target with rocket fire. The higher the skill level, the more likely they are to lead the target. If the target's feet are below the muzzle, then 2/3 of the time the tank/chick will fire at the target's feet. In all other cases they'll fire at the target's origin. Rocket speed is skill level dependent. On easy the default 500 units/sec is used. On nightmare it's a very speedy 800 units/sec. (Speed adjustments are not made if homing rockets are used).

Added **HOMING_ROCKETS** spawnflag to boss2, chick, supertank, tank, and tank_commander

Umm... whoops. Density calculations used for func_pushables were applied to **all** entities, and guess what? Monster_flipper is too heavy to swim :-). This has been fixed.

Another whoops. Previously if fog was active when the user changed levels with the "map" command, the fog remained active in the next map. The code now correctly turns fog off at map exit.

June 10, 2000

Added health value to **func_train**, **model_train**, and **model_spawn**. IOW these entities may now be shot and destroyed. Removed the ineffective "block shots" solid state from model_spawn and model_train, which never worked.

Added **target_rocks**, used to generate a landslide

Added several features to **target_blaster**: It may now be toggled like a target_laser. It can target any entity, moving or not, including players. It has a choice of weapons.

Target_laser can target players. It can pulse on/off without the use of a func_timer.

Added **homing rockets** to available weapons for target_blaster and turret_breach. These things are nasty, the way you like 'em.

June 7, 2000

Added **target_anger** and **target_monsterbattle**. These compliment the new **dmgteam** monster key. Also made many changes to **monster** code.

Added NO_STUPID_SPINNING, NO_DROPTOFLOOR, and SHOOTABLE spawnflags to all **pickup items**.

Improved impact physics for **func_vehicle**. Also made it possible to damage func_vehicle.

Changed trigger code so that sounds=3 (silent) works correctly. In normal Quake2, setting sounds=3 sets the trigger to play the nonexistent trigger1.wav. It **is** silent, since the wav file doesn't exist, but an annoying error message is displayed at map startup.

June 4, 2000

I really don't care for idiot programmers who provide a constant barrage of "new and improved" updates, so I'll try not to become one. Having said that, a quick test by several folks at Rust turned up 3 fairly obnoxious problems that in my opinion were

serious enough to justify an update. So here it is :-).

Bug Fixes

Fixed problem with **target_lock**. Maps may only have 1 target_lock with the HUD spawnflag set, for obvious reasons. However, in previous version if the player restarted a map after DYING (rather than using the map command), then the code erroneously counted the same target_lock again, reported an error, and destroyed the lock. This was a particularly stupid error in the example map, since the HUD spawnflag isn't even supported in this version... what can I say? I was tired. I suppose this wasn't caught earlier because it was not possible to die in Tremor (where the target_lock originated).

Fixed a really nasty error with the **crane**, or actually with **func_pushables** being transported by the crane. In the previous (and this) version, stacked waterborne crates go through many gyrations in an attempt to unstack themselves. Depending on map geometry this may not be possible, but the stubborn crates will still try. In the previous version if you attempted to lift the top crate while this was going on, the crate would float up through the crane_hook and continue out of the level, and the crane was then completely flummoxed. In this version the crate touch function now checks to see whether a crate is being influenced by a crane, and bails out if so.

Changed the **func_vehicle** bounding box to rotate the extents as the vehicle rotates. In other words the bounding box will always be the smallest box that contains the vehicle. In previous version, the bounding box was initially squared off, making the small dimension equal to the large dimension. This mostly worked but was overly conservative (too large). Its harder now to get the vehicle stuck, but unfortunately still not impossible. Mapmakers should only use func_vehicles in fairly simple squared-off areas.

Other changes

Changed the method for taking control of and disengaging from func_vehicle. To drive the vehicle, hop on and press the +use key once. To stop driving, press +use again. You will **not** be thrown from the vehicle as in the previous version.

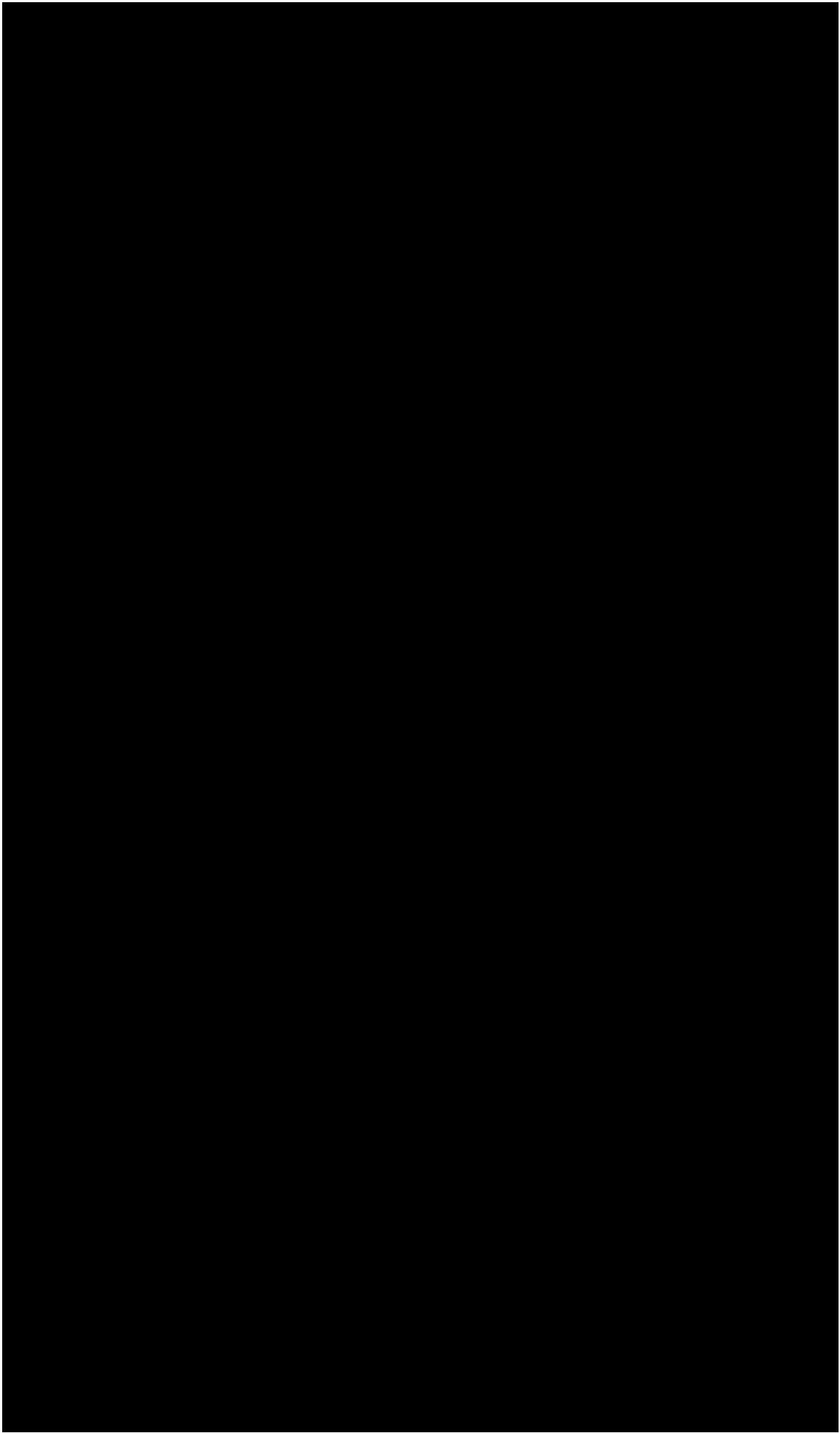
Func_vehicle now damages monsters and opposing player (if moving). I'm sure we'll be playing around with the damage parameters a bit.

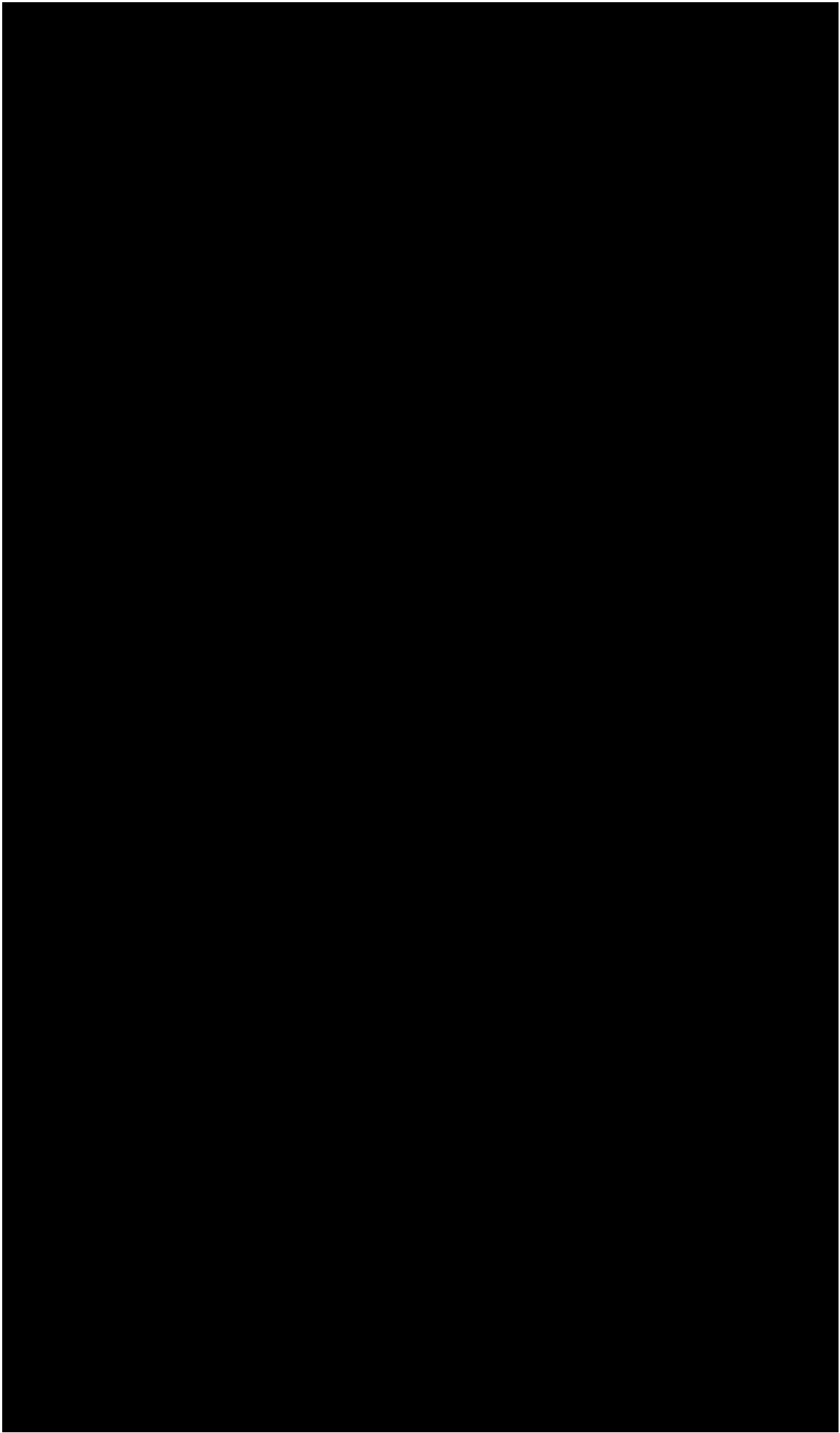
A couple of changes have been made to the example maps. In hyde2.bsp, I've added another crane_control that gives a better view of things (and illustrates that multiple crane_controls work :-)). In hyde3.bsp, the architecture has been changed a bit to make it harder to get the func_vehicle stuck.

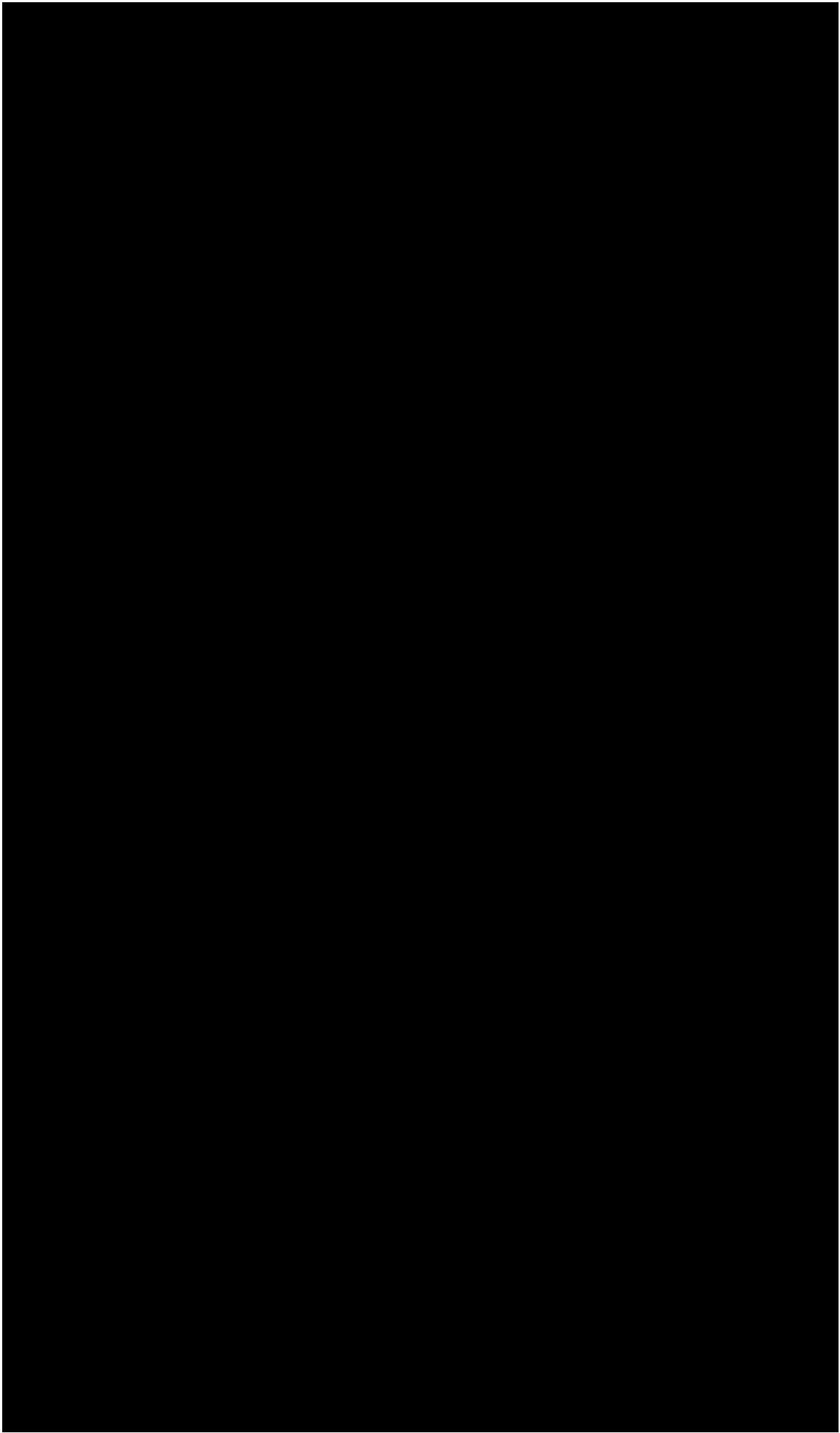
June 1, 2000

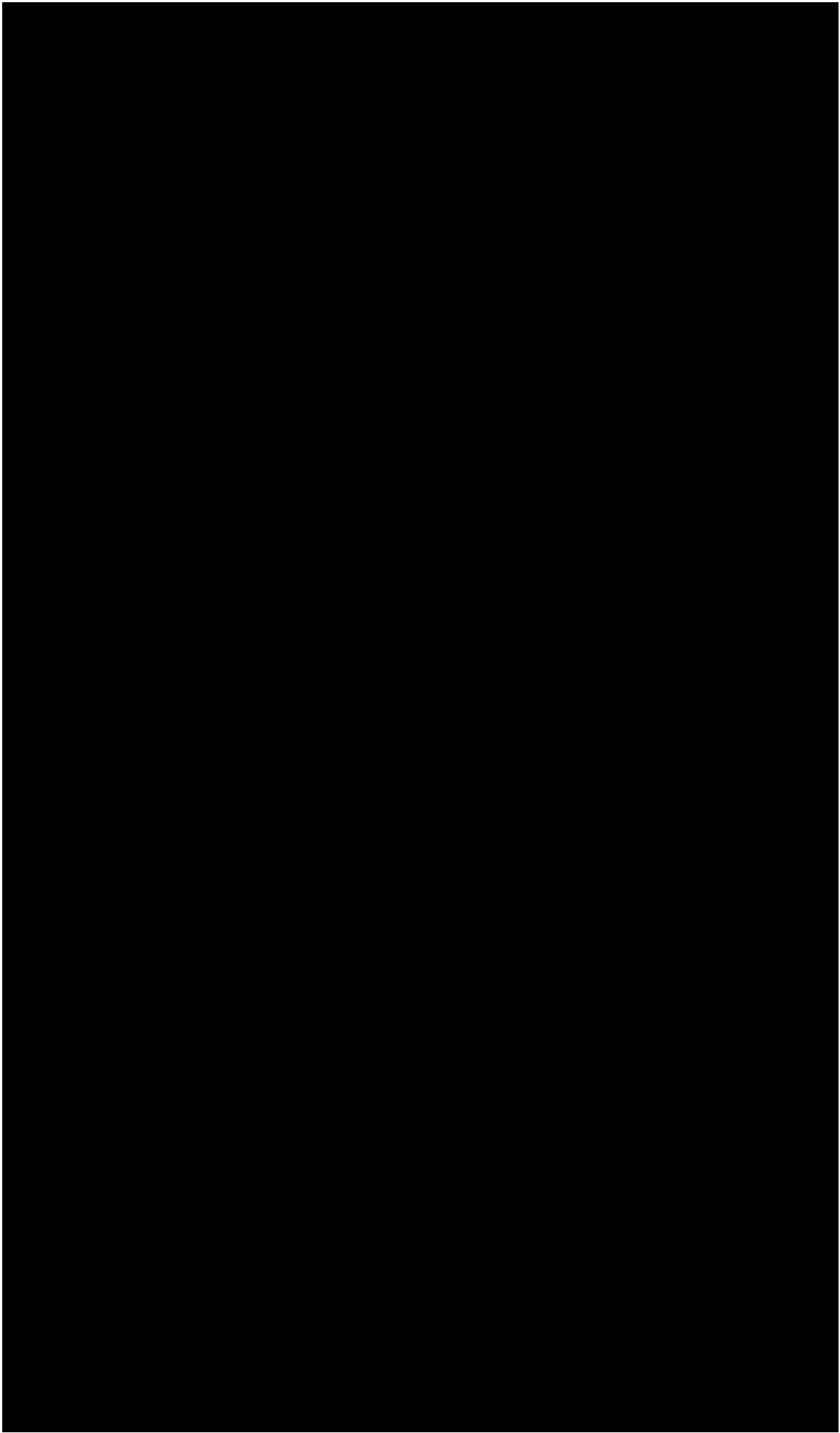
Initial release

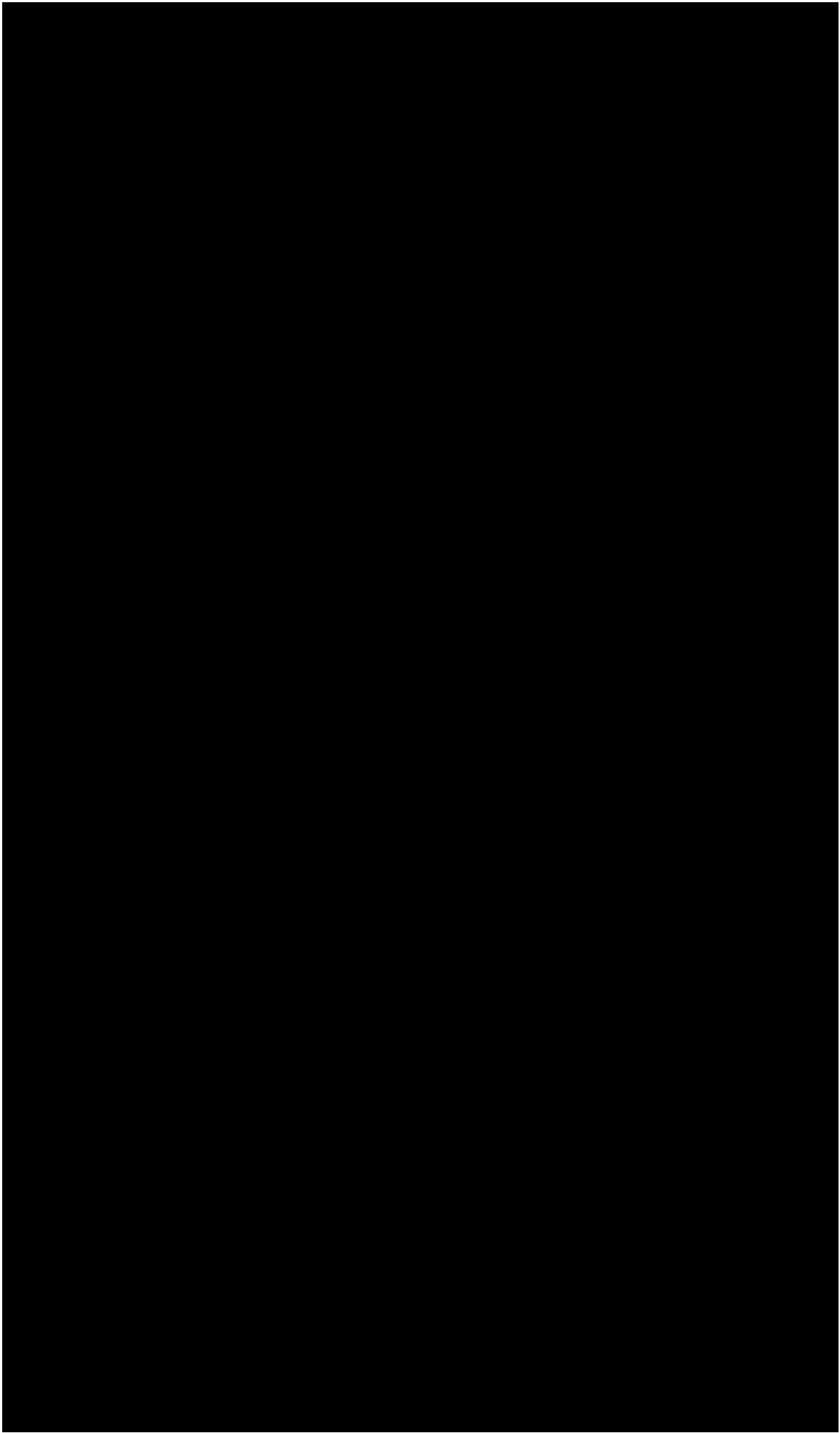
Lazarus Main Page

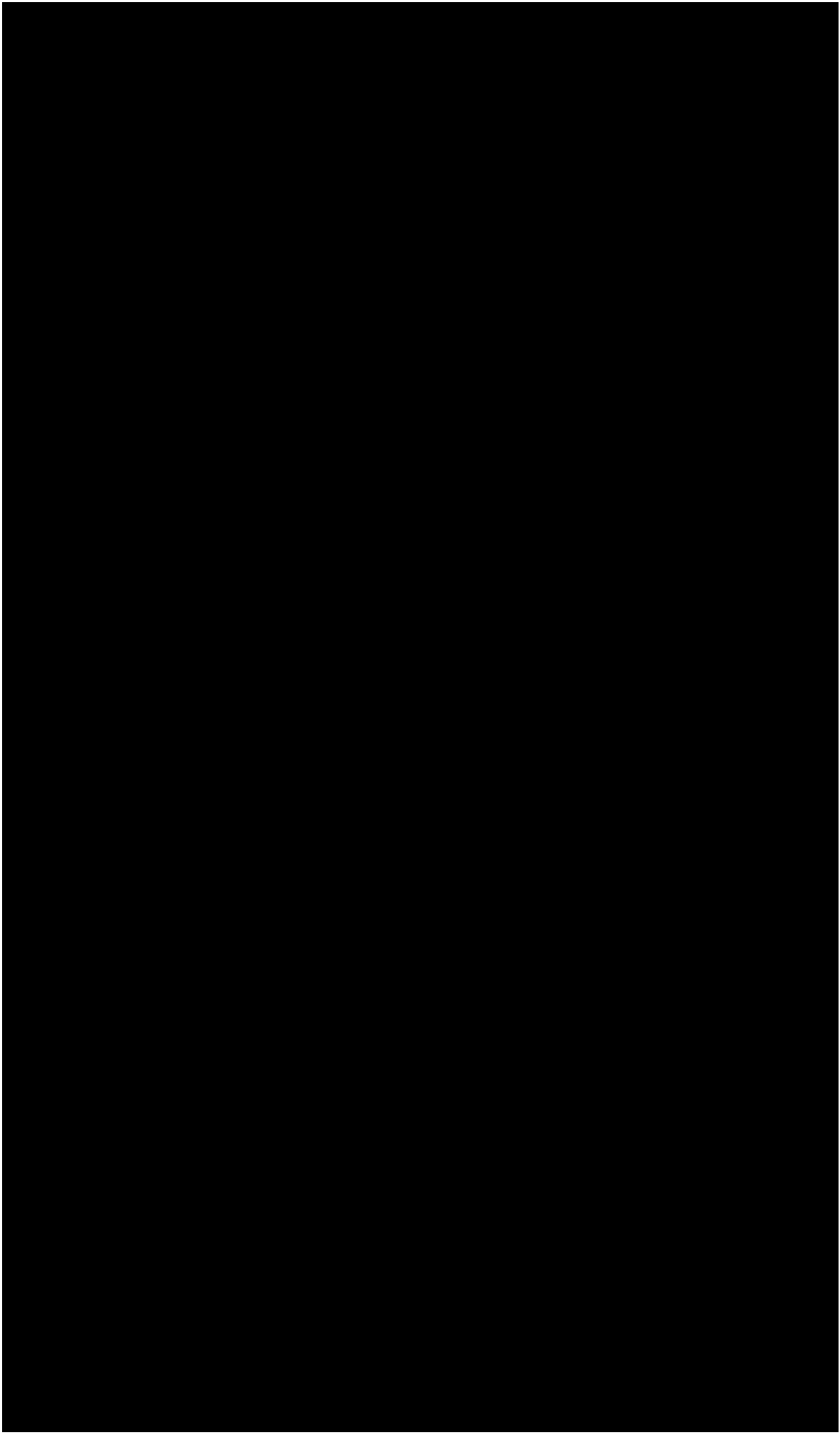


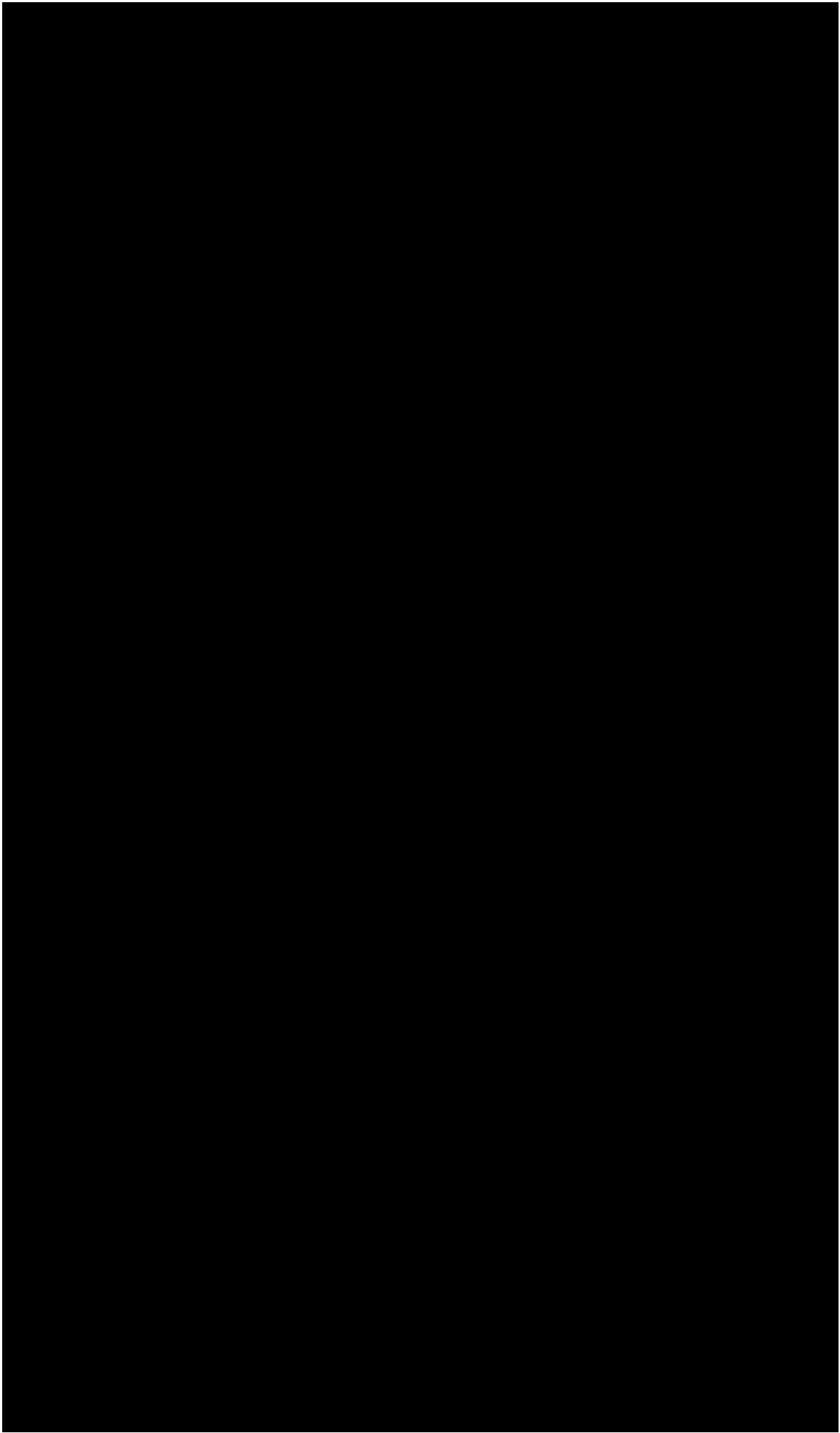


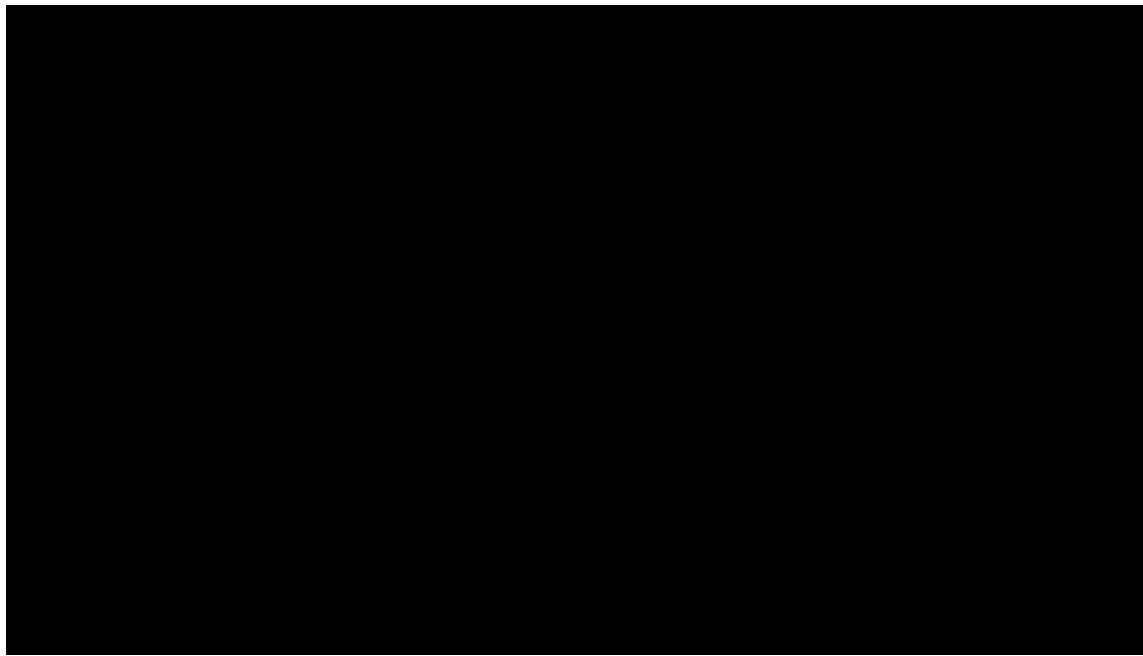












Lazarus Downloads



For the latest available files, please visit the [Lazarus Downloads](#) page at the website. You can sign up to be alerted to interim beta releases there as well.

[Lazarus Main Page](#)

Info_notnull

The **Lazarus** info_notnull is identical to the standard Quake2 info_notnull point entity, with the addition of **movewith** support. This allows it to move with its parent entity.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0. Only relevant if the info_notnull is used as a teleporter destination.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0. Only relevant if the info_notnull is used as a teleporter destination.

movewith

Targetname of the parent entity the info_notnull is to **movewith**.

targetname

Name of the specific info_notnull.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

Info_notnull will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Info_notnull will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Info_notnull will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Info_notnull will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Info_player_coop

The **Lazarus** info_player_coop is identical to the standard Quake2 info_player_coop point entity, with the addition of the **style** key, which specifies the player's starting weapon, if any. This setting will override a style setting used in **worldspawn**. By setting different style values to different coop starts, each player in a squad could be made to spawn with a unique weapon. Or by using multiple versions of the starts in the same places, and inhibiting them based on skill level, you could give the players decent weapons at the outset on easy skill, while denying that advantage in the harder skill levels.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0.

style

Specifies the player's starting weapon. Overrides the style setting used by **worldspawn**. If non-zero, the player's current inventory will be cleared; this will not serve to add a weapon to his current arsenal.

Choices are:

- 0: Don't specify (default)
- 1: Blaster
- 2: Shotgun, Blaster backup
- 3: Super Shotgun, Blaster backup
- 4: Machinegun, Blaster backup
- 5: Chaingun, Blaster backup
- 6: Grenade Launcher, Blaster backup
- 7: Rocket Launcher, Blaster backup
- 8: Hyperblaster, Blaster backup
- 9: Railgun, Blaster backup
- 10: BFG10K, Blaster backup
- 1: No weapon at all
- 2: Shotgun, no Blaster
- 3: Super Shotgun, no Blaster
- 4: Machinegun, no Blaster
- 5: Chaingun, no Blaster
- 6: Grenade Launcher, no Blaster
- 7: Rocket Launcher, no Blaster
- 8: Hyperblaster, no Blaster
- 9: Railgun, no Blaster
- 10: BFG10K, no Blaster

targetname

Name of the specific info_player_coop.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The info_player_coop will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The info_player_coop will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The info_player_coop will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The info_player_coop will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Info_player_deathmatch

The **Lazarus** info_player_deathmatch is identical to the standard Quake2 info_player_deathmatch point entity, with the addition of the **style** key, which specifies the player's starting weapon, if any. This setting will override a style setting used in **worldspawn**. By setting different style values to different deathmatch starts, players can spawn with a unique weapon depending on where they spawn in the map.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0.

style

Specifies the player's starting weapon. Overrides the style setting used by **worldspawn**. Choices are:

- 0: Don't specify (default)
- 1: Blaster
- 2: Shotgun, Blaster backup
- 3: Super Shotgun, Blaster backup
- 4: Machinegun, Blaster backup
- 5: Chaingun, Blaster backup
- 6: Grenade Launcher, Blaster backup
- 7: Rocket Launcher, Blaster backup
- 8: Hyperblaster, Blaster backup
- 9: Railgun, Blaster backup
- 10: BFG10K, Blaster backup
- 1: No weapon at all
- 2: Shotgun, no Blaster
- 3: Super Shotgun, no Blaster
- 4: Machinegun, no Blaster
- 5: Chaingun, no Blaster
- 6: Grenade Launcher, no Blaster
- 7: Rocket Launcher, no Blaster
- 8: Hyperblaster, no Blaster
- 9: Railgun, no Blaster
- 10: BFG10K, no Blaster

targetname

Name of the specific info_player_deathmatch.



Info_player_start

The **Lazarus** info_player_start is identical to the standard Quake2 info_player_start point entity, with the addition of the **style** and **health** keys. "Style" specifies the player's starting weapon, if any. This setting will override a style setting used in **worldspawn**. By using multiple player starts in the same place, and inhibiting them based on skill level, you could give the player a decent weapon at the outset on easy skill, while denying that advantage in the harder skill levels. "Health" is used to specify the player's health value at startup.

Also, this entity enjoys **count** support, which allows it to be auto-killtargeted. This feature is only available when deathmatch=0, and the info_player_start is not being used as an info_player_intermission.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0.

count

When non-zero, specifies the number of times the start will be used as a spawn point before being auto-killtargeted (see [this page](#) for details). Default=0. Ignored unless deathmatch=0.

health

If non-zero, health specifies an upper-end for the player's health when he spawns into the map at this location. If player's health was already lower than this value, he won't gain health points. This might be useful if the story of a map calls for (for example) the player crash-landing at the start of a map.

style

Specifies the player's starting weapon. Overrides the style setting used by **worldspawn**. If non-zero, the player's current inventory will be cleared; this will not serve to add a weapon to his current arsenal.

Choices are:

- 0**: Don't specify (default)
- 1**: Blaster
- 2**: Shotgun, Blaster backup
- 3**: Super Shotgun, Blaster backup
- 4**: Machinegun, Blaster backup
- 5**: Chaingun, Blaster backup
- 6**: Grenade Launcher, Blaster backup
- 7**: Rocket Launcher, Blaster backup
- 8**: Hyperblaster, Blaster backup
- 9**: Railgun, Blaster backup
- 10**: BFG10K, Blaster backup
- 1**: No weapon at all
- 2**: Shotgun, no Blaster
- 3**: Super Shotgun, no Blaster
- 4**: Machinegun, no Blaster
- 5**: Chaingun, no Blaster
- 6**: Grenade Launcher, no Blaster
- 7**: Rocket Launcher, no Blaster
- 8**: Hyperblaster, no Blaster

- 9: Railgun, no Blaster
- 10: BFG10K, no Blaster

targetname

Name of the specific info_player_start. When loading a map with multiple info_player_starts from the console, the start that the player will spawn at will be the first unnamed start found in the map file. If all starts are named, it will be the first start found.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

Info_player_start will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Info_player_start will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Info_player_start will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

Info_player_start will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Install & Run Lazarus



Lazarus is a *modification* to **Quake2** and as such cannot be run as a standalone game - you'll need to have **Quake2** installed first. In the event that you've never run a Quake2 mod before, here are some instructions to running a level using the **Lazarus** game code:

- Make sure **Quake2** is already installed and that you've patched it to the current, final version (v3.20).
- Unzip the files marked as "essential" on the [downloads](#) page into the same directory where you installed **Quake2** (this is the same folder on your machine where you'll find QUAKE2.EXE). Be sure to have the "Use Folder Names" or similar option checked for whatever file extraction program you're using.

Do *not* extract the files in the QUAKE2/BASEQ2 directory. If you do, the best thing that will happen is that you won't be able to get **Lazarus** to work; the worst thing that will happen is you'll have to reinstall and re-patch Quake2.

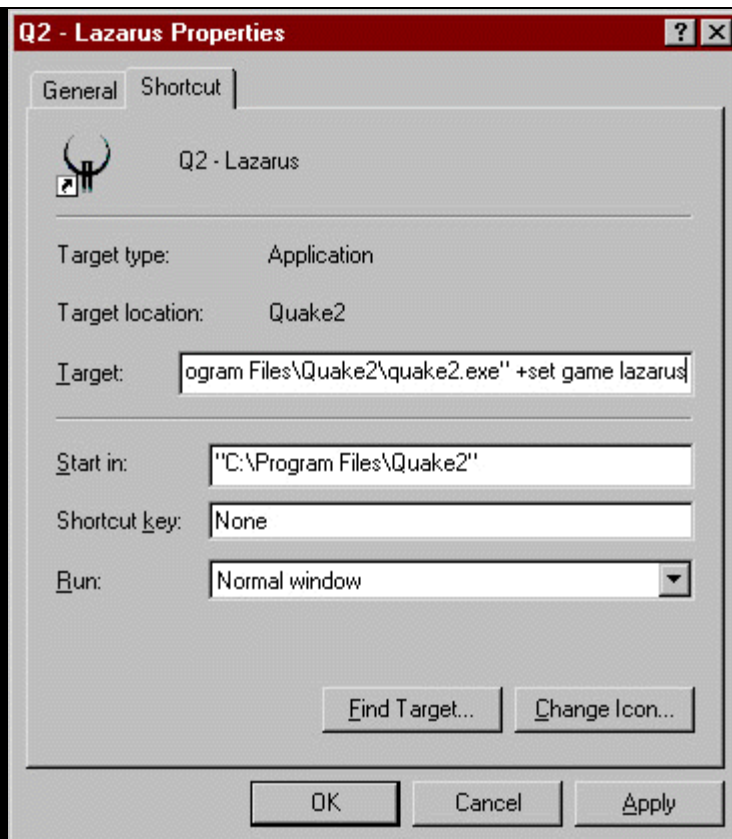
If you can't open .ZIP files, then you need a compressed file extraction utility. The most well-known one is [WinZip](#); the free shareware version will work fine.

- Right-click on QUAKE2.EXE and select *Create Shortcut*. Rename it to whatever you wish. Then right-click on the newly-created shortcut and select *Properties*. On the Properties dialog that results, go to the *Shortcut* tab and edit the command line in the *Target* field to include +set game lazarus, as in the following example (which was obviously renamed to Q2 - Lazarus):

Note that you must leave a space between the end of the command line and the beginning of the +set game lazarus part.

If your command line has long folder names and/or folder names with spaces (as in "Program Files" in the example), then your command line will be enclosed by quotation marks. The new information should be added **outside** the quotation marks, as shown.

If your command line doesn't have quotation marks, then don't worry about it.



- Move the shortcut to wherever you want and double-click on it to start up a **Q2 Lazarus** game. You should be brought immediately to the main menu; if you see the Q2 demos running then something is wrong.
- At this point you can load any map you wish (either from the **Lazarus** game folder, or any Q2 map from the standard game. Standard Q2 maps will not contain any of the new **Lazarus** entities of course, but new monster AI and behavior, the new physics tweaks, and access to the **Lazarus** sniper zoom be in effect.
- **Lazarus** offers a few new gameplay commands that need key assignments in order for you to use them. *These commands cannot be assigned in the "Configure Keys" menu, since there's no way for us to modify that menu.* Instead the commands must be assigned manually, either through the console or by loading a configuration file. A configuration file which accomplishes this is provided for you, and it may be loaded at any time by typing **LAZBINDS** at the console (the command isn't case-sensitive). The commands and their default key assignments are:

Command	Key Assignment
+use <i>This lets you push and pull objects, access remote turret controls, and operate manual triggers.</i>	ENTER
use Flashlight <i>Select and use the Flashlight.</i>	F
use Jetpack <i>Toggle the Jetpack on/off.</i>	J
+zoomin <i>Enters sniper mode when pressed; increases zoom magnification level when held.</i>	Z
+zoom <i>Enters sniper mode when pressed and held, at the last magnification level used. Exits sniper mode when released.</i>	X
+zoomout	C

Enters sniper mode when pressed; decreases zoom magnification level when held.

You can change these key assignments by editing the included file, *suggested.cfg*, and then re-entering the "LazBinds" command when in-game. If you find editing such files to be a stumbling block, then some info that may be of help is available [here](#). More info on **Lazarus**-specific console commands is found on [this page](#).

DISCLAIMER

Neither David Hyde nor Tony Ferrara make any warranties regarding this product. Users assume responsibility for the results achieved for computer program use and should verify applicability and accuracy.

Lazarus Main Page

Item_*

Lazarus adds a few notable enhancements to all `item_*` point entities, which can allow for their more realistic placement in the map as well as making them destroyable by weapon fire. And, **movewith** support is also included. The entities affected are:

- `item_adrenaline`
- `item_ancient_head`
- `item_armor_body`
- `item_armor_combat`
- `item_armor_jacket`
- `item_armor_shard`
- `item_bandolier`
- `item_breather`
- `item_enviro`
- `item_flashlight`
- `item_freeze`
- `item_health`
- `item_health_large`
- `item_health_mega`
- `item_health_small`
- `item_invulnerability`
- `item_jetpack`
- `item_pack`
- `item_power_screen`
- `item_power_shield`
- `item_quad`
- `item_silencer`

Please see the descriptions for the **NO_DROPTOFLOOR**, **NO_STUPID_SPINNING** and **SHOOTABLE** spawnflags for more details.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0. Ignored for spinning items unless **NO_STUPID_SPINNING** is set.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0. Ignored for spinning items unless **NO_STUPID_SPINNING** is set.

count

Only applies to the `item_flashlight` and `item_jetpack`.

For the flashlight: specifies the amount of cell ammo used per minute while the flashlight is in use.

Count=0 means the flashlight use is free. Flashlight count default=0.

For the jetpack: specifies the amount of fuel units contained in the jetpack when picked up. A negative count means the jetpack contains infinite fuel. Jetpack count default=0.

delay

Specifies the delay in seconds before **target**, **killtarget** and **message** will fire after the item is touched by

the player. Default=0.

health

Specifies hit points before the item will explode if the **SHOOTABLE** spawnflag is set. Default=20.

killtarget

Targetname of the entity to be removed from the map when the item is touched by the player.

message

Specifies the character string to print to the screen when the item is touched by the player.

movewith

Targetname of the parent entity the item is to **movewith**.

target

Targetname of the entity to be triggered when the item is touched by the player.

targetname

Name of the specific item entity.

team

Team name of the specific item entity. This is only relevant when deathmatch=1. When multiple pickups have identical team names, the first to appear in the .map file will be the one that appears on map load. This pickup entity serves as the master, and its teammates are the slaves. When the master pickup entity is picked up by a player, any of the teammates will spawn on a randomly rotating basis. Item entities may be teamed with other items, **ammo**, and/or **weapons**.

Spawnflags

NO_STUPID_SPINNING Spawnflag (=4)

The item_* entity will not spin (of course this doesn't affect items which don't spin to begin with). This allows proper use of the **angle** and **angles** keys. When used in conjunction with the **NO_DROPTOFLOOR** spawnflag, items that would normally spin may be placed in a realistic manner in your map.

NO_DROPTOFLOOR Spawnflag (=8)

The item_* entity will remain wherever it was placed in the map editor. This allows for placing the entity where it could be left hanging in the air, and/or it could be placed so that its bounding box intersects any solid brush, for example snugging the item up against a wall or laying it on its side. The normal 32x32x32 pickup "field" would still be in effect, so care must be taken so that the situation where the item can be picked up from the opposite side of a wall or similar is avoided. The item will not be lit if the entity's origin is buried in a world brush or otherwise hidden from light. The runtime "DropToFloor:" error diagnostics will not apply if this spawnflag is set. Please see **these tips** on making use of this feature easier.

SHOOTABLE Spawnflag (=16)

The item_* entity will be destroyable by weapon fire, as determined by **health**. The downside to using this option is that the item must be made solid. This is no problem if you can pick the item up... in the game, there will be a barely noticeable bump as you run into the model. But if you're already stocked up on the item, you won't be able to pick it up, **and** the item will block your path... not that this is a serious problem, but it's not what we're accustomed to. Naturally, destroying the item will clear the path, but the loss of the item could have consequences all its own. When deathmatch=1, a destroyed item entity will respawn in 30 seconds.

NOT_IN_EASY Spawnflag (=256)

The item_* entity will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The item_* entity will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The item_* entity will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The item_* entity will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Item_flashlight

The item_flashlight is a new **Lazarus** point entity, which works much like any other pickup item. But unlike other powerups, the flashlight doesn't expire. Instead, it uses 0 or more cells for power. Set the **count** key to the number of cells required per 1 minute usage.

To use the flashlight, bind a key to "use flashlight". (The **LazBinds** command will bind this command to the "F" key by loading **suggested.cfg**, which is included in the **main distribution**). So that mappers will not be required to distribute needless models/pics, the flashlight cannot be obtained with either the "give all" or "give flashlight" cheats, unless *developer* is set to 1.

You'll likely notice a couple of goofy visual effects concerning the flashlight: Brush models will be lit as though in their original position (at the time the map was compiled; Textures will be lit in proportion to the scale value they use. For example, a texture with a large scale will have a much larger lit area than a texture with a smaller scale.

As outlined in the **redistribution** doc, this entity will require the following files:

- [models/items/f_light/tris.md2](#)
- [models/items/f_light/skin.pcx](#)

Please refer to the **Item_*** page for a full list of all keys and spawnflags which can be used by the item_flashlight.

Key/value pairs

count

Specifies the amount of cell ammo used per minute when the flashlight is in use. When count=0 the flashlight use is free. Default=0.

Item_freeze

The item_freeze is a new **Lazarus** point entity which allows the player to, in effect, stop time. When used, monsters/actors stop in their tracks, projectiles stop in mid-air, and all time-dependent behavior ceases to function. You can't, for example, fire a weapon or touch triggers while the item_freeze is in use. So what's it good for? How about sneaking past a horde of monsters or getting yourself out of an extremely low-health situation?

To use item_freeze, bind a key to "use stasis generator". As with other Lazarus custom pickups, item_freeze cannot be obtained via cheat unless *developer* is set to 1, so that mappers will not be required to distribute needless models/pics.

Thanks to **venomus** for the item_freeze pickup model and sounds.

As outlined in the **redistribution** doc, this entity will require the following files:

- [models/items/stasis/tris.md2](#)
- [models/items/stasis/skin.pcx](#)
- [sound/items/stasis.wav](#)
- [sound/items/stasis_start.wav](#)
- [sound/items/stasis_stop.wav](#)

Please refer to the **Item_*** page for a full list of all keys and spawnflags which can be used by the item_freeze.

[Lazarus Main Page](#)

Item_jetpack

The item_jetpack is a new **Lazarus** point entity which allows the user to fly around a map. It requires fuel, so you might want to make some **ammo_fuel** pickups available in your map to go along with it. The default amount of fuel found in a jetpack is 0 units; but this may be changed by giving a value to **count**. When count=0, the jetpack may not be used until fuel is obtained. If count is negative, the jetpack does not consume fuel and may be used indefinitely.

To use the jetpack, bind a key to "use jetpack". (The **LazBinds** command will bind this command to the "J" key by loading **suggested.cfg**, which is included in the **main distribution**). Once the jetpack is activated, you may take off by using the "jump" (+moveup) command. Cancel upward thrust by releasing the "jump" key and you'll fall back to earth. Forward/back and strafing commands work the same while on the ground, but while in the air, they act as lateral thrusters. Speed in the air is not limited as speed on the ground is... if you have a lot of room you can accelerate to a nice speed. And if you hit the wall at too high of a speed, the result will be what you'd expect.

An active jetpack with no upward thrust applied will still apply some thrust. So you can use it to drop down from great heights, taking little or no damage when you land. The jetpack will shut itself down if you drop into the water. If you drop into lava or slime, it will blow up real good.

Fuel consumption varies. The jetpack will consume 1 unit of fuel per second simply by being on. It will guzzle an additional 10 units/second while thrust is being applied. Watch your fuel.

Two modes of jetpack use are available. There's the normal, you-have-to-work-to-stay-aloft, default behavior, and an easier "hover" version. Behavior is set with the **jetpack_weenie** cvar. This cvar is server-side only, to prevent players from getting an advantage over their opponents with the "fly cheat" version in a multiplayer game.

So that mappers will not be required to distribute needless models/pics, the jetpack cannot be obtained with either the "give all" or "give jetpack" cheats, unless *developer* is set to 1.

As outlined in the **redistribution** doc, this entity will require the following files:

- models/items/jetpack/tris.md2
- models/items/jetpack/skin.pcx
- pics/p_jet.pcx
- sound/jetpack/activate.wav
- sound/jetpack/rev.wav
- sound/jetpack/revrun.wav
- sound/jetpack/running.wav
- sound/jetpack/shutdown.wav
- sound/jetpack/stutter.wav

*The jetpack model distributed with Lazarus is courtesy of **ChaosDM**. The jetpack code is modified from source developed by Muce & Attila, available from **QDeveLs**. Jetpack sounds were created by jeffrey (Jeff Hayat) specifically for Lazarus.*

Please refer to the **Item_*** page for a full list of all keys and spawnflags which can be used by the item_jetpack.

Key/value pairs

count

Specifies the amount of fuel units contained in the jetpack when picked up. A negative count means the jetpack contains infinite fuel. Default=0.

Key_*

Lazarus adds a few notable enhancements to all key_* point entities, which can allow for their more realistic placement in the map as well as making them destroyable by weapon fire. Additionally, the **TRIGGER_SPAWN** and **NO_TOUCH** spawnflags, which only applied to the key_power_cube in the standard game, apply to all keys when run under **Lazarus**. And, **movewith** support is also included. The entities affected are:

- key_airstrike_target
- key_blue_key
- key_commander_head
- key_data_cd
- key_data_spinner
- key_pass
- key_power_cube
- key_pyramid
- key_red_key

Please see the descriptions for the **NO_DROPTOFLOOR**, **NO_STUPID_SPINNING** and **SHOOTABLE** spawnflags for more details.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0. Ignored for spinning keys unless **NO_STUPID_SPINNING** is set.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0. Ignored for spinning keys unless **NO_STUPID_SPINNING** is set.

delay

Specifies the delay in seconds before **target**, **killtarget** and **message** will fire after the key is touched by the player. Default=0.

health

Specifies hit points before the key will explode if the **SHOOTABLE** spawnflag is set. Default=20.

killtarget

Targetname of the entity to be removed from the map when the key is touched by the player.

message

Specifies the character string to print to the screen when the key is touched by the player.

movewith

Targetname of the parent entity the key is to **movewith**.

target

Targetname of the entity to be triggered when the key is touched by the player.

targetname

Name of the specific key entity.

Spawnflags

TRIGGER_SPAWN Spawnflag (=1)

The key_* entity must be called by another entity before it appears in the map. Unlike standard Quake2, this flag applies to all key entities and not just the key_power_cube.

NO_TOUCH Spawnflag (=2)

The key_* entity cannot be picked up by the player, and as a result it will exhibit a "solid" bounding box. Unlike standard Quake2, this flag applies to all key entities and not just the key_power_cube.

NO_STUPID_SPINNING Spawnflag (=4)

The key_* entity will not spin (of course this doesn't affect keys which don't spin to begin with). This allows proper use of the **angle** and **angles** keys. When used in conjunction with the **NO_DROPTOFLOOR** spawnflag, keys that would normally spin may be placed in a realistic manner in your map.

NO_DROPTOFLOOR Spawnflag (=8)

The key_* entity will remain wherever it was placed in the map editor. This allows for placing the entity where it could be left hanging in the air, and/or it could be placed so that its bounding box intersects any solid brush, for example snugging the key up against a wall or laying it on its side. The normal 32x32x32 pickup "field" would still be in effect, so care must be taken so that the situation where the key can be picked up from the opposite side of a wall or similar is avoided. The key will not be lit if the entity's origin is buried in a world brush or otherwise hidden from light. The runtime "DropToFloor:" error diagnostics will not apply if this spawnflag is set. Please see **these tips** on making use of this feature easier.

SHOOTABLE Spawnflag (=16)

The key_* entity will be destroyable by weapon fire, as determined by **health**. The downside to using this option is that the key must be made solid. In the game, there will be a barely noticeable bump as you run into the model.

NOT_IN_EASY Spawnflag (=256)

The key_* entity will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The key_* entity will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The key_* entity will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The key_* entity will be inhibited and not appear when deathmatch=1.

Light

The **Lazarus** light is identical to the standard Quake2 light point entity, with the addition of **count** support, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

_color

Specifies light color in relative RGB values. Default=1 1 1.

_cone

Specifies size of spotlight cone arc, in degrees. Range is 10-90. Default=10. Ignored if no **target** set.

count

When non-zero, specifies the number of times the light will be turned off before it is auto-killtargeted (see [this page](#) for details).

light

Specifies light brightness level. Default=300.

style

Specifies lightstyle type. Default=0. (See the [target_lightramp](#) page for a list of coded style values). Ignored when deathmatch=1.

target

Targetname of the entity the light aims at. If the light has a target set, it is a spotlight.

targetname

Name of the specific light. Lights with targetnames are present as entities at runtime, and are therefore edicts. Additionally, each uniquely named light is considered as a unique lightstyle during the radiosity compile process.

Spawnflags

START_OFF Spawnflag (=1)

Sets the light to be inactive when the map loads.

NOT_IN_EASY Spawnflag (=256)

Light will be inhibited and not appear when skill=0. Only relevant for lights with a **targetname**.

NOT_IN_NORMAL Spawnflag (=512)

Light will be inhibited and not appear when skill=1. Only relevant for lights with a **targetname**.

NOT_IN_HARD Spawnflag (=1024)

Light will be inhibited and not appear when skill=2 or greater. Only relevant for lights with a **targetname**.

Misc_actor

Misc_actor is a combatant that uses a player model. This entity was never fully implemented in the original Q2 code. **Lazarus** gives you the intended misc_actor functionality and adds several enhancements, including a choice of models and weapons, and the ability to turn an actor into a player-hunting monster, a player-defending ally, or a remote-controlled robot. Actors are not confounded by low obstacles - they'll simply jump right over them. They'll jump up a maximum of 48 units, and will jump down off ledges as far as 160 units. To prevent them from looking too silly, we've limited things so they will not jump on surfaces which have a steeper angle than 30 degrees from the horizontal, and they will not jump on players, monsters, or other actors. Starting with version 2.0, actors don't mind getting their feet wet, unlike other monsters. They'll enter water up to their eyeballs, but no deeper (they still don't know how to swim).

As currently implemented, misc_actors do not automatically attack anything (unless the **GOOD_GUY** or **BE_A_MONSTER** spawnflags are set). Instead, you direct the actions of actors by sending them to a **target_actor**; targeting them directly with a **target_anger**, or by making them play a series of animation frames with a **target_animation**.

On map load, **GOOD_GUY** actors standing in the vicinity of regular monsters will not attack them, nor will the monsters attack the GOOD_GUY actors. This is so they don't depopulate maps before the player gets a chance to explore them. GOOD_GUYS can get involved with what's going on in either of two ways: If you "+use" an actor (look at the actor and press the +use key), he'll follow you around and is now a monster-hunting killing machine. This behavior is toggled with the +use key - press it again to have an actor stand in place. If you do not activate an actor in this way, he'll still come to your defense when needed. Rather than wait for the player to take damage before attacking the player's enemy, they respond to any monster who gets mad at the player (assuming the GOOD_GUY actor is in the PVS of the player). Once all player enemies in the vicinity are eliminated, the GOOD_GUY actor will follow the player around. Since this means that the actor could potentially get underfoot, players have the ability to correct this with an active **go** command. Looking at a GOOD_GUY actor and entering "go" at the console (or alternatively pressing a key bound to "go") will tell the actor to "get out of the way".

GOOD_GUY actors following the player can be a little problematic, since they can't do everything the player can do, like leaping across gaps. This will very probably improve in the future, but even now the misc_actor is far from a total lamer. He can **jump onto and over obstacles, and can leap down from ledges** (see above). Of course, it's extremely doubtful that actions like pressing buttons will ever be something that a misc_actor can do. Keep all this in mind when designing areas of your map where GOOD_GUY actors are supposed to be. If for any reason you wish to disable actor jumping for your map, it can be easily done with the **actorjump** console variable, which can be set in the copy of default.cfg you would include with your map distribution.

For more in-depth information on how to set up your actor entity, please refer to the notes at the end of this page, which cover setting up the actor entity by selecting the **player model**, selecting the **skin**, customizing the actor's **bounding box**, and equipping the actor with **weapons**. You might also like to have a look at the **AI page**, since actor properties and behavior can change under certain conditions.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the facing angle of the actor on the XY plane. Default=0.

angles

Specifies the facing angle of the actor in 3 dimensions as defined by pitch, yaw, and roll. Default=0 0 0.

bleft

Specifies bottom-left (minY/minX/minZ) bounding box coordinates for the actor relative to the actor's origin. If no value is specified for either bleft or **tright**, and the value of **usermodel** is a player model in

the **Lazarus ActorPak**, then the suggested bounding box as listed on the ActorPak page for that model will be used. See **Bounding Box** notes for a full explanation.

combattarget

Targetname of the point_combat the actor will move to when angered.

deathtarget

Targetname of the entity to be triggered upon the actor's death.

distance

Specifies the maximum distance in map units that the actor can be from the player or other enemy and will still see and shoot at him. Minimum value is 500 (values<500 are reset to 500 by the code). Default=1280.

dmgteam

An actor with a dmgtteam value set will treat all attacks upon other actor(s) or monster(s) with the same dmgtteam value as an attack on himself. Also see [this page](#). Dmgteam may also be used to trigger a [trigger_relay](#).

flies

Specifies the probability that the actor's corpse will produce flies. When they appear, they will have a duration of 60 seconds before they expire. Flies will never appear if the actor's corpse is in a liquid. Normal probability values range from 0.00-1.00; 0.00=0% probability (never produce flies); 1.00=100% probability (always produce flies). If flies>1, the actor will have flies buzzing about him even while alive. Default=0. Ignored if **health**<0, unless flies=1. In that event, the initially dead actor will have flies which never expire.

gib_health

Specifies the number of hit points required to gib the actor's corpse. Ignored when **NO_GIB** is set. Default=40.

health

Specifies the number of hit points required to kill the actor.
If health=0 it will be reset to the default.
If health<0 the actor will be dead on startup.
If health=100000 or more the actor is invulnerable, and will ignore attacks.
Default=100.

item

Classname of the entity to be spawned by the actor upon his death. As with Lazarus monsters, actors may drop any health item on death.

killtarget

Targetname of the entity to be removed from the map upon the actor's death.

mass

Weight of the actor. Default=200.

move_origin

This vector (x y z) offset is only meaningful if the actor is the target of a **func_monitor**. When the player accesses a func_monitor that targets this actor, his viewpoint will be offset from the actor origin by this vector. X is in the forward direction, positive Y is to the left, and Z is vertical. If move_origin is left blank or is "0 0 0", then the player's viewpoint will be at the actor's *viewheight* (22 units above the origin).

movewith

Targetname of the parent entity the actor is to **movewith**.

muzzle

Specifies weapon firing origin, offset from the actor origin. See also **muzzle2**. Player models with visual

weapons (vwep) capabilities will normally require a weapon-specific muzzle offset. If not set, this value defaults to (18.4 7.4 9.6), which is roughly correct for the standard O2 male with a machinegun but not much else. If not set and **usermodel** is a player model in the **Lazarus ActorPak**, then the suggested muzzle value as listed on the ActorPak page for that model *and weapon* (as specified by **sounds**) will be used. Of course, if you want to set your own weapon firing origin, you are free to do so, and a number of muzzle-position-tweaking, in-game, **developer tools** are available for this purpose.

muzzle2

Specifies an optional 2nd weapon firing origin, offset from the actor origin. See also **muzzle**. Some player models (e.g. Walker and Rhino) carry two weapons - setting this key lets both weapons fire. If left blank or set to 0 0 0, the code assumes that the actor only carries a single weapon. Damage produced by an actor with two weapons is roughly equal to that of a single-weapon actor. For the blaster, machinegun, chaingun, and hyperblaster, both weapons will fire during every frame, each dispensing half the normal damage. All other weapons fire alternately between the 2 muzzle origins. If not set and **usermodel** is a player model in the **Lazarus ActorPak**, then the suggested muzzle2 value (if any) as listed on the ActorPak page for that model *and weapon* (as specified by **sounds**) will be used. Of course, if you want to set your own weapon firing origin, you are free to do so, and a number of muzzle-position-tweaking, in-game, **developer tools** are available for this purpose.

powerarmor

Specifies amount of optional power shield armor the actor has. Default=0.
Experiment with settings using the weapons the actor will face - it looks odd for a actor to die before his power shield runs out.

sounds

Specifies the weapon the actor will carry. A backup (close-range) weapon can also be specified here. See **Weapons** notes for details. Default=0 (no weapon).

style

Specifies the 0-based index of the skin the actor will use. Default=0. The valid range for style is 0 to the number of skins referenced from within the **model** itself minus one. What number references what skin is dependent on the individual model. If the style value +1 exceeds the number of referenced skins, skin 0 will be used. The number of available skins for all player models distributed with Lazarus are given in the **ActorPak** documentation. For info on how to select skins for the standard id player models, please refer to these **notes**.

target

Targetname of the path_corner the actor will move to.

targetname

Name of the specific actor.

tright

Specifies top-right (maxY/maxX/maxZ) bounding box coordinates for the actor relative to the actor's origin. If no value is specified for either tright or **bleft**, and the value of **usermodel** is a player model in the **Lazarus ActorPak**, then the suggested bounding box as listed on the ActorPak page for that model will be used. See **Bounding Box** notes for a full explanation.

usermodel

Specifies the player model used. Value is the name of the model's "player" folder. No default is specified for usermodel. If the model folder contains custom sounds, misc_actor will use them; otherwise it will fall back to using male model sounds. See **Player Model** notes for more information.

Spawnflags

AMBUSH Spawnflag (=1)

Identical to **SIGHT**, except that the code will not call any idle sounds. At this time the actor has no idle sounds, so currently this distinction is meaningless.

TRIGGER_SPAWN Spawnflag (=2)

The actor won't appear in the map until his targetname is called. If the actor has **BE_A MONSTER** set, he will not appear in the monster count tally on map load, but will appear once he is actually spawned.

SIGHT Spawnflag (=4)

The actor won't attack until angered or he has line-of-sight to his enemy. This is the same as normal Quake2's AMBUSH flag. Currently this flag is meaningless for actors that do not also have the **BE_A_MONSTER** flag set.

GOOD_GUY Spawnflag (=8)

The actor will treat all attacks upon the player as an attack on himself, and will attack any monster that he can see who gets mad at the player. GOOD_GUYS will follow the player around if there is no one to attack. GOOD_GUY's respond to player "+use" commands by moving out of the player's way. Takes priority over **BE_A_MONSTER**, but if the player shoots a GOOD_GUY/BE_A_MONSTER actor, the GOOD_GUY flag is removed and the actor will turn on the player.

NO_GIB Spawnflag (=16)

The actor cannot be gibbed. Visual evidence is that the corpse produces sparks rather than blood when shot.

HOMING Spawnflag (=32)

The actor uses homing rockets instead of standard rockets. This flag is ignored if neither of the actor's **weapons** is a rocket launcher. Homing rockets are somewhat slower than normal rockets, which actually makes them deadlier since they have more time to correct themselves.

BE_A_MONSTER Spawnflag (=64)

The actor is a monster, and will attack the player on sight, unless **GOOD_GUY** is also set. Monster actors appear in the monster/kill count tallies.

IGNORE_FIRE Spawnflag (=128)

The actor can shoot other actors/monsters without them getting mad at him. In practical terms, for this flag really only applies to **BE_A_MONSTER** actors. This is useful flag to set for a actor whose placement frequently causes him to hit other actors/monsters with friendly fire. Setting this flag does **not** mean that IGNORE_FIRE actors and **GOOD_GUY** actors/ **monsters** won't shoot each other.

NOT_IN_EASY Spawnflag (=256)

The actor will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The actor will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The actor will be inhibited and not appear when skill=2 or greater.

Notes

Player Model

Misc_actors can use any player model at all. Which model is to be used is defined by the **usermodel** key. The catch is that every entity in the game that uses a model needs the model's skins to be referenced in the model itself. The exception to this rule is the player model, and therefore the player models that come with the game do not reference any skins. However this exception leaves the misc_actor stuck using a model without any skins.

So, you can't use standard player models out of the box. But never fear! If you'd like to use the standard Q2 male, female, or cyborg models, **Lazarus** will generate modified versions of them, which **do** reference skins, on the fly. This also means that you won't have to distribute them along with your maps (nor should you, since that would violate id Software's EULA). If your map uses a misc_actor which calls a standard model, when the user runs your map, the new modified model will be generated on the user's machine. This means no copyright violations for you, a smaller download for the user, and best of all we have the blessing of id Software for this (thanks Paul!). *(Auto-generating model code courtesy of Argh!).*

Now that you've set the model, now you need to set the skin. So, read on...

Skin

Selecting the proper skin is done by assigning a value to the misc_actor's **style** key. As noted under the style key description, the valid range for style is 0 to the number of skins referenced from within the player model itself minus one. So each model will be different. The player models distributed with Lazarus have been already modified to include references to all possible skins (which are themselves included). The **ActorPak** documentation includes instructions for making similar changes to models not distributed with Lazarus, so if we haven't included your pet player model, **and** you've done the Right Thing™ and have secured the model author's permission to redistribute his work, you can make different player models compatible with misc_actor.

If you're using the Q2 male, female, or cyborg model, then use the following table to determine the **style** value to select your desired skin from among the standard Q2 skins. (Just as you do not need to redistribute the id player models, you also do not need to redistribute the skins. The user will have them already, in his \BASEQ2\ folder, and they'll work fine from there). If you're using a custom skin, rename it to "customXX", and select a style value that references a custom skin. Each id model as generated by **Lazarus** allows for 32 different skins.

<u>style value</u>	<u>male</u>	<u>female</u>	<u>cyborg</u>
0	cipher	athena	oni911
1	claymore	brianna	ps9000
2	flak	cobalt	tyr574
3	grunt	ensign	custom1
4	howitzer	jezebel	custom2
5	major	jungle	custom3
6	nightops	lotus	custom4
7	pointman	stiletto	custom5
8	psycho	venus	custom6
9	rampage	voodoo	custom7
10	razor	custom1	custom8
11	recon	custom2	custom9
12	scout	custom3	custom10
13	sniper	custom4	custom11
14	viper	custom5	custom12
15-31	custom1-17	custom6-22	custom13-29

Bounding Box

The original misc_actor used the normal Q2 player model, and therefore used the normal player's bounding box. **Lazarus** allows the use of any player model, and many of the custom player models are either much smaller or larger than the Q2 male, and/or possess animation sequences that cause them to move well outside of that normal player bounding box. While this may be OK in a deathmatch game, there's no reason not to use more realistic values which are appropriate for the player model you are using.

The **bleft** and **tright** keys allow you to specify a custom-sized bounding box for each individual misc_actor. Finding the optimum bounding box for a given player model is a kind of compromise. On the

one hand, it's generally desirable to devise a bounding box large enough to contain all of the animation frames, but on the other hand it's not desirable to make the bounding box so large that the actor will take damage when shot in what appears to be only empty space around and/or above him. Basically, you want to find the smallest bounding box that still avoids the appearance of the actor clipping into walls, or that his head intersects with overhead obstacles. If shots appear to pass harmlessly through the body (not the extremities) of the actor, it's a safe bet the bounding box is too small. Use the **bbox** console command to fine-tune the bounding box, if it becomes necessary.

If no bounding box coordinates are specified, and the value of **usermodel** is a player model included with the **Lazarus ActorPak**, then bleft and tright will assume values which will build the bounding box suggested for that model on the ActorPak page. If no bounding box coordinates are specified, and the value of **usermodel** is something other than an ActorPak model, then bleft and tright will assume values which will build the standard player bounding box.

Weapons

Actors can be given any of the weapons available to the player. But, since actors use the monster AI code, and will sometimes run right up to the player in order to attack him, some additional coding has been added to avoid the effect of an actor getting in his enemy's face and then blowing himself up with the rocket launcher. After all, the actor (just like the player) has no melee attack.

If the actor's weapon is an explosive weapon, he will stop approaching his enemy when he comes within 200 units of him. He won't back off and attempt to maintain that distance if his enemy approaches him, however.

An actor can also be given a secondary weapon. If he has one, he won't stop at the 200-unit point; instead, once he is within 200 units of his enemy, he'll switch to his secondary weapon. If the distance grows to 300 units or more, he'll switch back to his primary weapon again. So, think of an actor equipped with 2 weapons as one who has a ranged (primary) weapon, and a close-combat (secondary) weapon. A good example would be an actor with a rocket launcher as a primary weapon and a super shotgun as a secondary weapon. Of course you can equip your actors however you please. If you want to give an actor a blaster as a primary weapon and let him go kamikaze with a BFG at close range, hey it's your party.

An actor can also have a secondary weapon, while having no primary weapon specified. Why would you want to do this? Because such an actor will not fire until he closes to within 200 units. At that point he'll whip out his secondary weapon and start firing. To avoid the silly effect of the weapon disappearing again once the actor is more than 300 units away, the code considers such an actor's secondary weapon as his primary weapon after it becomes active.

Actors will always have their primary weapons active at map load. Specifying what the primary and secondary weapons are is done using the **sounds** key. How this is done gets a bit complicated, so pay attention :-)

$$\text{Sounds} = (\text{primary weapon code} * 100) + (\text{secondary weapon code}).$$

Weapon codes are what you would expect: 0=none, 1=blaster, 2=shotgun, etc. So sounds=703 means that the primary weapon is the rocket launcher and the secondary weapon is the super shotgun. If sounds is less than 100, the lack of a secondary weapon is assumed, and the sounds value is used as the primary weapon. As an example, sounds=7 is equivalent to sounds=700).

This covers all possibilities save one: how to give an actor a secondary weapon but no primary weapon. To do this, use the negative value of the weapon code. For example, sounds=-3 means the actor has no primary weapon and uses the super shotgun as his secondary weapon.



Misc_deadsoldier

The **Lazarus** misc_deadsoldier is identical to the standard Quake2 misc_deadsoldier point entity, with the addition of the ubiquitous **movewith** support, the ability to use a variety of (bloodless) male player skins by means of the **style** value, and the **FLIES** spawnflag. This last offers the option of creating the pleasant atmosphere of having flies buzzing around your dead guys, in the same manner as often seen with enforcer and mutant corpses. When the dead guy is gibbed, the flies disappear.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies facing angle on the XY plane. Default=0.

angles

Specifies facing angle in 3 dimensions, defined as pitch, yaw, and roll. Default=0 0 0.

movewith

Targetname of the parent entity the dead guy is to **movewith**.

style

The style value allows you to specify an alternate skin to use with misc_deadsoldier to add a bit of variety to your map. Note that since the normal male player model does not include pain skins, the misc_deadsoldier will be bloodless for all but the default skin, which looks at best a bit odd with headless soldiers.

0: Default	8: Pointman
1: Cipher	9: Psycho
2: Claymore	10: Rampage
3: Flak	11: Razor
4: Grunt	12: Recon
5: Howitzer	13: Scout
6: Major	14: Sniper
7: Nightops	15: Viper

targetname

Name of the specific dead guy.

Spawnflags

ON_BACK Spawnflag (=1)

Dead guy is lying on his back.

ON_STOMACH Spawnflag (=2)

Dead guy is lying on his stomach.

BACK_DECAP Spawnflag (=4)

Dead guy is headless and lying on his back.

FETAL_POS Spawnflag (=8)

Dead guy is curled in the fetal position.

SIT_DECAP Spawnflag (=16)

Dead guy is headless and sitting.

IMPALED Spawnflag (=32)

Dead guy is on his back with limbs hanging.

FLIES Spawnflag (=64)

Dead guy has flies buzzing around him.

NOT_IN_EASY Spawnflag (=256)

Dead guy will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Dead guy will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Dead guy will be inhibited and not appear when skill=2 or greater.

[Lazarus Main Page](#)

Misc_easter*

The **only** change made to misc_easterchick1, misc_easterchick2, and misc_eastertank is the ability to make these entities non-solid. Why is this important? For the chicks the importance is debatable, but for a fake tank riding a moving, rotating brush model that the player can also ride, his large bounding box will often inconveniently intersect the player's bounding box even though the models don't actually touch. Due to a bit of convoluted logic in the physics code, this situation can cause the player to block the brush model, which usually results in hurting the player (in this case for no apparent reason). A non-solid misc_eastertank is used in the lcraft **example map** for this very reason - if the tank were solid the player would be hurt while riding the helicopter simply by being near the tank as the helicopter rotates.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Spawnflags

NON_SOLID Spawnflag (=1)

If set, the entity is not solid.

NOT_IN_EASY Spawnflag (=256)

The monster will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The monster will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The monster will be inhibited and not appear when skill=2 or greater.

[Lazarus Main Page](#)

Misc_explobox

Lazarus modifies the standard Quake2 misc_explobox with the addition of the **BARREL_GIBS** spawnflag. If set, the entity uses custom debris models rather than the standard debris. This option is not set automatically because it adds 5 new models to a map, so if used you'll need to pay attention to the total number of models used (which cannot exceed 256).

As outlined in the **redistribution** doc, if **BARREL_GIBS** is set, misc_explobox requires these files:

- models/objects/barrel_gibs/barrel_gib1.md2
- models/objects/barrel_gibs/barrel_gib2.md2
- models/objects/barrel_gibs/barrel_gib3.md2
- models/objects/barrel_gibs/barrel_gib4.md2
- models/objects/barrel_gibs/barrel_gib5.md2
- models/objects/barrel_gibs/skin.pcx

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the initial direction on the XY plane. Default=0.

angles

Specifies the initial direction in 3 dimensions, defined by pitch and yaw (roll is ignored). Default=0 0 0.

dmg

Specifies the amount of damage hit points the misc_explobox will generate when it is destroyed. Default=150.

health

Damage sustained by the misc_explobox before destruction. Default=10

mass

The weight of misc_explobox. Heavier barrels are pushed slower. Default=400.

Spawnflags

BARREL_GIBS Spawnflag (=1)

If set, custom debris models are used when misc_explobox detonates.

NOT_IN_EASY Spawnflag (=256)

Misc_explobox will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Misc_explobox will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Misc_explobox will be inhibited and not appear when skill=2 or greater.

Misc_insane

The **Lazarus** misc_insane is nearly identical to the standard Quake2 misc_insane point entity, with the additions of **health**, **gib_health**, and **mass** keys. These give the options to set the insane guy's hit points for both death and gibbing, and also set his weight. We've also added the effect of a misc_insane's corpse producing flies sometimes, in the same manner as enforcers and mutants in the standard game. This probability of this happening is set with the **flies** key.

Legend:

Compared to Standard Quake2, Key/Flag is:

New

Modified

Unchanged

Key/value pairs

angle

Specifies the facing angle of the insane guy on the XY plane. Default=0.

angles

Specifies the facing angle of the insane guy in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0.

combattarget

Targetname of the point_combat the insane guy will move to when triggered.

deathtarget

Targetname of the entity to be triggered upon the insane guy's death.

flies

Specifies the probability that the insane guy's corpse will produce flies. When they appear, they will have a duration of 60 seconds before they expire. Flies will never appear if the insane guy's corpse is in a liquid. Normal probability values range from 0.00-1.00; 0.00=0% probability (never produce flies); 1.00=100% probability (always produce flies). If flies>1, the insane guy will have flies buzzing about him even while alive. Default=0.30. Ignored if **health**<0.

gib_health

Specifies the number of hit points required to gib the insane guy's corpse. Default=50.

health

Specifies the number of hit points required to kill the insane guy. Default=100.

item

Classname of the entity to be spawned by the insane guy upon his death.

killtarget

Targetname of the entity to be removed from the map upon the insane guy's death.

mass

Weight of the insane guy. Default=300.

movewith

Targetname of the parent entity the insane guy is to **movewith**.

target

Targetname of the path_corner the insane guy will move to.

targetname

Name of the specific insane guy.

Spawnflags

AMBUSH Spawnflag (=1)

Insane guy won't move to his point_combat until he sees the player, or until the player injures him.

TRIGGER_SPAWN Spawnflag (=2)

Insane guy won't appear in the map until his targetname is called.

CRAWL Spawnflag (=4)

Insane guy crawls instead of walks.

CRUCIFIED Spawnflag (=8)

Insane guy is in an immobile crucified position.

STAND_GROUND Spawnflag (=16)

Insane guy will stand in place and never walk around.

ALWAYS_STAND Spawnflag (=32)

Insane guy will never drop to his knees.

NOT_IN_EASY Spawnflag (=256)

Insane guy will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Insane guy will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Insane guy will be inhibited and not appear when skill=2 or greater.

Lazarus Main Page

Misc_strogg_ship

Lazarus modifies the standard Quake2 misc_strogg_ship by making it damageable if its **health** value is greater than 0. It also gives the misc_strogg_ship the same rotational abilities as the **func_train**. For positive health values, misc_strogg_ship is solid and can be damaged by any weapons fire. When "killed", the ship explodes using the same routine used by func_explosive. **Dmg** and **mass** values determine the damage radius and number and size of debris particles generated by the explosion. You may optionally trigger another event when the ship is killed by setting the **deathtarget** value.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the initial ship direction on the XY plane. Default=0.

angles

Specifies the initial ship direction in 3 dimensions, defined by pitch and yaw (roll is ignored). Default=0 0 0.

deathtarget

Specifies the entity to trigger when the ship is destroyed. Not used if health=0.

dmg

Specifies the amount of damage hit points the misc_strogg_ship will generate when it is destroyed. Ignored if health=0. Default=200.

health

Damage sustained by the misc_strogg_ship before destruction. Default=0 (invulnerable)

mass

Determines the number and size of debris particles generated when ship is killed. Ignored if health=0. Default=800.

pitch_speed

The rotational speed around the ship's lateral Y axis, in degrees/second, that ROTATE misc_strogg_ships will use while turning up or down. The current pitch_speed may be changed at each path_corner in the sequence by setting the **path_corner's pitch_speed**. Note that pitch_speed does not have a default value, so if you want your misc_strogg_ship to pitch up or down you **must** supply a value here.

roll_speed

The rotational speed around the ship's longitudinal axis, in degrees/second, that ROTATE misc_strogg_ships will use while turning toward the next **path_corner's roll** value. The current roll_speed may be changed at each path_corner in the sequence by setting the **path_corner's roll_speed**. Note that roll_speed does not have a default value, so if you want your misc_strogg_ship to roll you **must** supply a value here.

speed

Specifies the speed of the misc_strogg_ship, in units/sec. Default=300. Speed may be changed at each path_corner to the path_corner's speed value.

target

Targetname of the first path_corner in a path_corner sequence that the misc_strogg_ship will follow.

targetname

Name of the specific misc_strogg_ship.

yaw_speed

The rotational speed around the Z axis, in degrees/second, that ROTATE misc_strogg_ships will use while turning. The current yaw_speed may be changed at each path_corner in the sequence by setting the **path_corner's yaw_speed**. Note that yaw_speed does not have a default value, so if you want your misc_strogg_ship to rotate around the Z axis you **must** supply a value here.

Spawnflags**START_ON Spawnflag (=1)**

If set, misc_strogg_ship begins its movement when the map loads.

ROTATE (=8)

If set, the misc_strogg_ship will turn towards the next path_corner using its current **yaw_speed** and **pitch_speed** values (degrees/second).

SMOOTH_MOVE Spawnflag (=128)

If set, misc_strogg_ship will skip the last 0.1 second slowdown before reaching a path_corner so that it maintains a constant velocity. For more information see the description for **SMOOTH_MOVE** in the func_train reference.

Lazarus Main Page

Misc_teleporter

The **Lazarus** misc_teleporter adds quite a few changes to the standard Quake2 misc_teleporter point entity. First, the **movewith** key is added, which allows it to move with its parent entity. Then a number of new spawnflags are also added, which makes the teleporter a much more flexible entity. The **START_OFF** and **TOGGLE** flags support teleporters that are only functional some of the time, and the **NO_MODEL** flag lets the mapper do away with the teleporter model and effects entirely.

The **LANDMARK** flag allows the teleporting player to inherit his viewing angles and velocity he had at the time he teleported. These inherited view angles can still be modified by the angles value of the destination, if desired. LANDMARK teleporters support the optional use of the **trigger_transition**.

Finally, the **MONSTER** flag allows monsters to use the teleporter as well. (Of course, the trick here is in getting monsters to the teleporter in the first place).

Also see the **trigger_teleporter**, which is a brush model which may be used as an alternative to the misc_teleporter.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Facing angle of the teleporter model on the XY plane. Default=0.

angles

Facing angle of the teleporter model in 3 dimensions as defined by pitch, yaw, and roll. Default=0 0 0.

movewith

Targetname of the parent entity the teleporter is to **movewith**.

target

Targetname of the entity the teleporter will use as a destination.

targetname

Name of the specific teleporter.

Spawnflags

START_OFF Spawnflag (=1)

Sets the teleporter to be inactive when the map loads.

TOGGLE Spawnflag (=2)

Allows the teleporter to be toggled on/off.

NO_MODEL Spawnflag (=4)

The teleporter base model will not appear, the particle effects will not be rendered, and the teleporter idle sound will not be played.

MONSTER Spawnflag (=8)

The teleporter can transmit monsters and actors.

LANDMARK Spawnflag (=64)

The teleporter will cause the player's view angles and velocity to be preserved when he spawns at his destination. View angles are rotated by the angles value of the destination entity minus the inherited angles. If this flag is set, a **trigger_transition** may be used.

NOT_IN_EASY Spawnflag (=256)

The teleporter will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The teleporter will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The teleporter will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The teleporter will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Misc_teleporter_dest

The **Lazarus** misc_teleporter_dest is identical to the standard Quake2 misc_teleporter_dest point entity, with the addition of **movewith** support. This allows it to move with its parent entity.

Legend:

Compared to Standard Quake2, Key/Flag is:

New

Modified

Unchanged

Key/value pairs

angle

Specifies the facing angle of the teleporter_dest's model on the XY plane. Also sets the facing angle of the player, monster, or actor which spawns at it. Default=0.

angles

Specifies the facing angle of the teleporter_dest's model in 3 dimensions as defined by pitch, yaw, and roll. Also sets the facing angle in 3 dimensions of the player, monster, or actor which spawns at it. Default=0 0 0.

movewith

Targetname of the parent entity the teleporter_dest is to **movewith**.

targetname

Name of the specific teleporter_dest.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The teleporter_dest will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The teleporter_dest will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The teleporter_dest will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The teleporter_dest will be inhibited and not appear when deathmatch=1.

Misc_viper

Lazarus modifies the standard Quake2 misc_viper by making it damageable if its **health** value is greater than 0. It also gives the misc_viper the same rotational abilities as the **func_train**. For positive health values, misc_viper is solid and can be damaged by any weapons fire. When "killed", the viper explodes using the same routine used by func_explosive. **Dmg** and **mass** values determine the damage radius and number and size of debris particles generated by the explosion. You may optionally trigger another event when the viper is killed by setting the **deathtarget** value.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the initial viper direction on the XY plane. Default=0.

angles

Specifies the initial viper direction in 3 dimensions, defined by pitch and yaw (roll is ignored). Default=0 0 0.

deathtarget

Specifies the entity to trigger when the viper is destroyed. Not used if health=0.

dmg

Specifies the amount of damage hit points the misc_viper will generate when it is destroyed. Ignored if health=0. Default=200.

health

Damage sustained by the misc_viper before destruction. Default=0 (invulnerable)

mass

Determines the number and size of debris particles generated when viper is killed. Ignored if health=0. Default=800.

pitch_speed

The rotational speed around the viper's lateral Y axis, in degrees/second, that ROTATE misc_vipers will use while turning up or down. The current pitch_speed may be changed at each path_corner in the sequence by setting the **path_corner's pitch_speed**. Note that pitch_speed does not have a default value, so if you want your misc_viper to pitch up or down you **must** supply a value here.

roll_speed

The rotational speed around the viper's longitudinal axis, in degrees/second, that ROTATE misc_vipers will use while turning toward the next **path_corner's roll** value. The current roll_speed may be changed at each path_corner in the sequence by setting the **path_corner's roll_speed**. Note that roll_speed does not have a default value, so if you want your misc_viper to roll you **must** supply a value here.

speed

Specifies the speed of the misc_viper, in units/sec. Default=300. Speed may be changed at each path_corner to the path_corner's speed value.

target

Targetname of the first path_corner in a path_corner sequence that the misc_viper will follow.

targetname

Name of the specific misc_viper.

yaw_speed

The rotational speed around the Z axis, in degrees/second, that ROTATE misc_vipers will use while turning. The current yaw_speed may be changed at each path_corner in the sequence by setting the path_corner's yaw_speed. Note that yaw_speed does not have a default value, so if you want your misc_viper to rotate around the Z axis you **must** supply a value here.

Spawnflags

START_ON Spawnflag (=1)

If set, misc_viper begins its movement when the map loads.

ROTATE (=8)

If set, the misc_viper will turn towards the next path_corner using its current yaw_speed and pitch_speed values (degrees/second).

SMOOTH_MOVE Spawnflag (=128)

If set, misc_viper will skip the last 0.1 second slowdown before reaching a path_corner so that it maintains a constant velocity. For more information see the description for SMOOTH_MOVE in the func_train reference.

Lazarus Main Page

Misc_viper_bomb

Lazarus modifies the standard Quake2 misc_viper_bomb point entity by making it re-usable, if the new **MULTI_USE** spawnflag is set. And, with the addition of **count** support, this entity may be optionally auto-killtargeted after a specified number of uses.

Even better is the way **Lazarus** improves the way the viper bomb references its velocity and trajectory. In the standard game, a misc_viper triggers a path_corner pathtarget, which in turn triggers the viper bomb. The speed and direction of the misc_viper is used to give the viper bomb its speed and trajectory. Unfortunately, every viper bomb in a given map will use the speed and direction of the first misc_viper found in that map. If you wanted vipers traveling in different directions and at different speeds to each drop their own bombs and have those bombs drop in different directions and speeds as well, you'd be out of luck. Now it's possible.

This is done by pointing the misc_viper_bomb to a unique owner, typically a misc_viper, by using the **pathtarget** key. This key tells the bomb to use the speed and direction of that pathtargeted entity. Since every bomb can have a different reference pathtarget, every bomb can have a unique velocity and trajectory. This change allows the further modification where it is now possible to drop bombs without using a moving entity at all as an owner - for this you could have the misc_viper_bomb pathtarget itself. If it pathtargets itself, then it uses itself as a reference - and you can set the bomb's **speed** and **angle** internally. You could optionally use **angles** in lieu of angle, and give the bomb an upward initial trajectory - and thereby put mortars in your map.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the initial bomb trajectory on the XY plane. Default=0. Ignored if the **pathtarget** value is not equal to the value of the viper bomb's **targetname**.

angles

Specifies the initial bomb trajectory in 3 dimensions, defined by pitch and yaw (roll is ignored). Default=0 0 0. Ignored if the **pathtarget** value is not equal to the value of the viper bomb's **targetname**.

count

When non-zero, specifies the number of times the misc_viper_bomb will be called before being auto-killtargeted (see [this page](#) for details). Default=0. Ignored when **MULTI_USE** is not set.

dmg

Specifies the amount of damage hit points the misc_viper_bomb will generate at its origin. Default=1000.

killtarget

Targetname of the entity to be removed from the map when the viper bomb explodes.

pathtarget

Targetname of the entity the viper bomb will use as a speed and trajectory reference. The viper bomb may pathtarget itself and therefore use itself as a reference; in that event **speed** and **angle/angles** should be given meaningful values. If no pathtarget is set, the first misc_viper found in the map will be used as the reference entity.

speed

Specifies the initial bomb speed in units/second. Default=0. Ignored if the **pathtarget** value is not equal to the value of the viper bomb's **targetname**.

target

Targetname of the entity to be triggered when the viper bomb explodes.

targetname

Name of the specific misc_viper_bomb.

Spawnflags**MULTI_USE Spawnflag (=1)**

Specifies that the misc_viper_bomb may be triggered more than once.

NOT_IN_EASY Spawnflag (=256)

The misc_viper_bomb will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The misc_viper_bomb will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The misc_viper_bomb will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The misc_viper_bomb will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Miscellaneous Features



Zoom

A fully adjustable sniper mode allows you to smoothly zoom in/out between the default field-of-view and 5 degrees (or anywhere in between). It is also possible to snap instantly between normal and the last used sniper setting as well. The relevant commands are:

- **+zoomin** - when pressed, sniper mode is active and fov will decrease smoothly until the key is released or the fov is 5 degrees.
- **+zoomout** - when pressed, sniper mode is active and fov will increase smoothly until the key is released or the fov is equal to the default fov.
- **+zoom** - toggles the view between the default fov and the last fov set when **+zoomin** or **+zoomout** keys were released.
- **zoomrate** - persistent cvar controlling the speed of **+zoomin** and **+zoomout**. Default value is 80 degrees/second.
- **zoomsnap** - cvar specifying the fov to snap to with **+zoom**. Initially 20 degrees. This value is overwritten every time **+zoomin** or **+zoomout** are released (unless a **+zoomout** operation takes you to the default fov).

More **sniper zoom** commands can be found on the [console commands](#) page. Default bindings for the above sniper commands can be found on [this page](#).

The crosshair used by Lazarus is a slightly modified version of one created by Madman. You can check more of his and others' crosshair creations at www.bullseyecrosshairs.com.

Note: Lazarus adjusts mouse and joystick sensitivity while zooming to give the player roughly the same control he has with the normal fov. This feature comes at a price, however: Zoom is disabled for multiplayer games. There's really no good way to determine or change a client's mouse and joystick sensitivities without adding a bandwidth burden. As a result we've decided to leave this feature out of multiplayer games for now.

Homing rockets

Target_blaster, **turret_breach**, and rocket-firing **monsters** and **actors** can fire rockets that home in on a target, adjusting their path in flight. If you play around with these rockets (see **blaster.map** and **turret.map** in the examples) you'll likely notice that slower rockets (say 400 units/sec) tend to be deadlier than rockets that fly at the default speed (650 for the player, 500 for monsters), simply because they have more time to adjust their direction.

If the rocket's target is a player, the correction factor applied to it's path is skill level dependent. On easy, you'll probably be able to jump over rockets without taking too much damage. On nightmare... forget about it. Depending on speed, the rocket will likely be able to follow your jump.

In my (always correct) opinion, if you use homing rockets against monsters in your map it had better be a hellacious (technical term) battle, or it will be just plain unfair. Monsters don't know about or understand homing rockets, and they will be your basic sitting ducks.

Note: Homing rockets aim for the center of the bounding box of the target entity. If the target is, for example,

a func_train or func_explosive with nothing but air at the center, the rocket will pass straight through the target and turn, but most likely **never** hit the target.

Gib fade

This isn't an earth-shattering change, but we thought we should bring it to your attention just so you'd be sure to notice it. Ever get annoyed with the game making gibs go "POOF" after 10 seconds or so? No? Well nevermind then :-). But next time you run a map in Lazarus, be sure to stick around and watch the gibs fade away. Total gib duration isn't changed, but during the last 2 seconds they'll become progressively more transparent before drifting off to Strogg heaven.

Third-person perspective

Third-person perspective as implemented in Lazarus was originally developed by **SKULL**. Lazarus has one addition: If *tpp_auto* is set, Lazarus will automatically switch the player's view to a third-person perspective when he starts pushing/pulling a func_pushable, and switch back to the normal view when he loses contact with the func_pushable (unless *tpp* was already on). The player may also toggle third person view with the *thirdperson* console command.

There are a few problems with third-person perspective: It works reasonably well in open areas, but in tight spots it is.... not good. In tight spots the camera is prone to move into solid brushes... it usually corrects itself sooner or later, but the problem is definitely noticeable. In some cases (when the player is in a particularly tight spot) the view goes nuts - hopping back and forth attempting to find a "good" viewpoint. There's really not a good way around these problems other than designing maps around it (in other words don't build tight spots).

[Lazarus Main Page](#)

Model_spawn

This feature uses MapPack's model_spawn entity from the version 3 source code, with quite a few changes to get it working correctly. Model_spawn allows you to place an arbitrary model inside your map at any location, and optionally add a whizbang environmental effect to its appearance.

Key/Value pairs

bleft

the point that is at the bottom left of the models bounding box in a model editor.

effects

1: ROTATE Rotate like a weapon
 2: GIB
 8: BLASTER Yellowish orange glow plus particles
 16: ROCKET Rocket trail
 32: GRENADE Grenade trail
 64: HYPERBLASTER BLASTER w/o the particles
 128: BFG Big green ball
 256: COLOR_SHELL
 512: POWERSCREEN Green power shield
 16384: FLIES Ewww
 32768: QUAD Blue shell
 65536: PENT Red shell
 131072: TELEPORTER Teleporter particles
 262144: FLAG1 Red glow
 524288: FLAG2 Blue glow
 1048576: IONRIPPER
 2097152: GREENGIB
 4194304: BLUE_HB Blue hyperblaster glow
 8388608: SPINNING_LIGHTS Red spinning lights
 16777216: PLASMA
 33554432: TRAP
 67108864: TRACKER
 134217728: DOUBLE Yellow shell
 268435456: SPHERETRANS Transparent
 536870912: TAGTRAIL
 1073741824: HALF_DAMAGE
 2147483648: TRACKER_TRAIL

framenumbers

The number of frames you want to display after startframe

health

If non-zero **and** solidstate is 3 or 4 (solid), the entity will be shootable. When destroyed, it blows up with a no-damage explosion.

movewith

Targetname of the entity to move with

renderfx

1: MINLIGHT Never completely dark
 2: VIEWERMODEL
 4: WEAPONMODEL
 8: FULLBRIGHT
 16: DEPTHACK

32: TRANSLUCENT Transparent
64: FRAMELERP
128: BEAM
512: GLOW Pulsating glow of normal Q2 pickup items
1024: SHELL_RED
2048: SHELL_GREEN
4096: SHELL_BLUE
32768: IR_VISIBLE
65536: SHELL_DOUBLE
131072: SHELL_HALF_DAMAGE White shell
262144: USE_DISGUISE

skinnum

Specifies the 0-based index of the skin (.pcx file) to use on the model. Default=0.

solidstate

1. 1 - not solid at all. These models do not obey any sort of physics. If you place them up in the air or embedded in a wall they will stay there and be perfectly happy about it.
2. 2 - solid. These models will "droptofloor" when spawned. If the health value is set they may be damaged.
3. 3 - solid. Same as above but not affected by gravity. Model will remain in the same location.
4. 4 - not solid but affected by gravity. Model will "droptofloor" when spawned.

NOTE : if you want the model to be solid and/or drop to the floor then you must enter vector values for **bleft** and **tright**.

startframe

The starting frame : default 0

style

Specifies the animation type to use.

0: None (unless startframe and framenumbers are used)

1: ANIM01 - cycle between frames 0 and 1 at 2 hz

2: ANIM23 - cycle between frames 2 and 3 at 2 hz

3: ANIM_ALL - cycle through all frames at 2 hz

4: ANIM_ALLFAST - cycle through all frames at 10 hz

Note: The animation flags override **startframe** and **framenumbers** settings you may have entered. ANIM_ALL and ANIM_ALLFAST don't do what you might think - rather than setting a framerate, these apparently cause the model to cycle through its ENTIRE sequence of animations in 0.5 or 0.1 seconds... which for monster and player models is of course not possible. Looks exceptionally goofy.

tright

the point that is at the top left of the models bounding box in a model editor

usermodel

The model to load ("models/" will be tacked on to the front, or if **PLAYER** is set, "players/" will be tacked on to the front and "/tris.md2" to the end).

Spawnflags:

TOGGLE (=2)

Start active, when triggered become inactive

PLAYER (=8)

Set this if you want to use a player model. If **usermodel** is blank, model_spawn will use the model of the player. Otherwise the model found at players/<usermodel>/tris.md2 will be used. Note that if you specify a player model using usermodel, it **must** be a model that has been modified as described in the **ActorPak** documentation, or the model will not be skinned.

NO_MODEL (=16)

Don't use a model. Useful for placing particle effects and dynamic lights.

ANIM_ONCE Spawnflag (=32)

Each time the model_spawn is triggered, it runs through one animation cycle and shuts off. TOGGLE will automatically be set if ANIM_ONCE is set.

[Lazarus Main Page](#)

Model_train

Model_train combines most of the features of **model_spawn** with **func_train**, allowing you to move md2 models around your map. Other than the obvious, there are at least two benefits to this capability: 1) You can create a window model, give it the RF_TRANSLUCENT renderfx and/or EF_SPHERETRANS effect, and team moving transparent windows with your func_doors, func_trains, etc. 2) You can use model_spawn's NO_MODEL spawnflag coupled with one of the rendering effects and simulate a moving spotlight. Settings are for the most part identical to those of model_spawn, of course with the addition of func_train parameters. There are a few differences in the behavior of this entity from a func_train that you should be aware of:

- The model_train follows a series of path_corners just as a func_train does, but unlike a func_train the model_train is placed such that it's origin is at the path_corner, rather than it's minimum bounding box coordinates. You'll need to know where the origin of the md2 model is to be able to correctly place the model_train.
- The lighting of md2 models, unlike brush models, is dependent on the location of the model. Stated more correctly, the model is lit according to the location of its origin. This is generally a **good** thing, but can cause unexpected results. For example, if your window model has its origin at the center, and the door that the window is part of closes into a wall such that the origin of the window is hidden, the entire window will become dark as soon as the window origin passes into the wall.

Key/Value pairs

bleft

The point that is at the bottom left of the model's bounding box in a model editor

effects

1: ROTATE Rotate like a weapon
2: GIB
8: BLASTER Yellowish orange glow plus particles
16: ROCKET Rocket trail
32: GRENADE Grenade trail
64: HYPERBLASTER BLASTER w/o the particles
128: BFG Big green ball
256: COLOR_SHELL
512: POWERSCREEN Green power shield
16384: FLIES Ewwwww
32768: QUAD Blue shell
65536: PENT Red shell
131072: TELEPORTER Teleporter particles
262144: FLAG1 Red glow
524288: FLAG2 Blue glow
1048576: IONRIPPER
2097152: GREENGIB
4194304: BLUE_HB Blue hyperblaster glow
8388608: SPINNING_LIGHTS Red spinning lights
16777216: PLASMA
33554432: TRAP
67108864: TRACKER
134217728: DOUBLE Yellow shell
268435456: SPHERETRANS Transparent
536870912: TAGTRAIL
1073741824: HALF_DAMAGE
2147483648: TRACKER_TRAIL

health

If non-zero **and** solidstate is set to SOLID_BBOX, the entity will be shootable. When destroyed, it blows up with a no-damage explosion.

noise

Audio file to play while the model_train is moving.

pitch_speed

roll_speed

yaw_speed

For **ROTATE** trains, pitch_speed and yaw_speed are the rotational velocities of the train while turning towards the next **path_corner**. Roll_speed is the rotational speed while rolling to the next path_corner's **roll** value. Once the train faces the next path_corner, yaw and pitch rotations stop. Likewise, once the train's roll angle is equal to the next path_corner's roll value, roll rotation stops. There are no default values for these speeds, so if you want the func_train to turn towards a path_corner and/or roll, you **must** supply non-zero values for the corresponding rotational speeds. Conversely, if you want your func_train to change yaw but not pitch for path_corners at different elevations, set pitch_speed to 0.

For a **ROTATE_CONSTANT** train, these are the constant rotational values that will be used by the func_train until either the train is toggled off or a path_corner changes the value. As with ROTATE trains, these speeds may be changed at every path_corner. **However**, for ROTATE_CONSTANT trains the path_corner values have a different meaning. Rather than specifying absolute values, the path_corner values are the **change** in speed. This was done so that 0 or unspecified values at path_corners have no effect, but you can still stop rotation by using the negative of the current rotational speed.

Finally, for a **SPLINE** train, roll_speed is never used, and pitch_speed and yaw_speed are **only** used to prevent rotation in a given direction by setting to a value less than 0. For example, if you don't want a train to change its pitch angle even though the path_corners comprising the train's route are at different elevations, set pitch_speed=-1.

renderfx

- 1: MINLIGHT Never completely dark
- 2: VIEWERMODEL
- 4: WEAPONMODEL
- 8: FULLBRIGHT
- 16: DEPTH HACK
- 32: TRANSLUCENT Transparent
- 64: FRAMELERP
- 128: BEAM
- 512: GLOW Pulsating glow of normal Q2 pickup items
- 1024: SHELL_RED
- 2048: SHELL_GREEN
- 4096: SHELL_BLUE
- 32768: IR_VISIBLE
- 65536: SHELL_DOUBLE
- 131072: SHELL_HALF_DAMAGE White shell
- 262144: USE_DISGUISE

skinnum

Specifies the 0-based index of the skin (.pcx file) to use on the model. Default=0.

solidstate

1. SOLID_NOT - not solid at all. These models do not obey any sort of physics. If you place them up in the air or embedded in a wall they will stay there and be perfectly happy about it.
2. SOLID_BBOX - solid. Unlike model_spawn, though, this setting will not cause model_train to "droptofloor". If the health value is a positive number the model_train may be damaged.

NOTE : if you want the model to be solid then you must enter vector values for **bleft** and **tright**.

startframe

The frame to display (ignored if **style**=3 or 4): default 0

framenumbers

The number of frames you want to display after startframe

style

Specifies the animation type to use.

0: None

3: ANIM_ALL - cycle through all frames at 2 hz

4: ANIM_ALLFAST - cycle through all frames at 10 hz

target

Targetname of the first path_corner on the model_train's route.

targetname

Name of the model_train. Unnamed model_trains will always start on, regardless of the **START_ON** setting.

team

Team name of the model_train. Trains with an identical team name will move together, and will all stop if one is blocked.

tright

The point that is at the top left of the models bounding box in a model editor

usermodel

The model to load ("models/" will be tacked on to the front, or if **PLAYER** is set, "players/" will be tacked on to the front and "/tris.md2" to the end).

Spawnflags:

START_ON Spawnflag (=1)

If set, model_train begins its movement when the map loads.

TOGGLE Spawnflag (=2)

If set, allows the train to be toggled on/off multiple times.

BLOCK_STOPS Spawnflag (=4)

The model_train will stop completely when an entity is blocking its way.

PLAYER (=8)

Set this if you want to use a player model. If **usermodel** is blank, model_train will use the model of the player. Otherwise the model found at players/<usermodel>/tris.md2 will be used. Note that if you specify a player model using usermodel, it **must** be a model that has been modified as described in the **ActorPak** documentation, or the model will not be skinned.

NO_MODEL (=16)

Don't use a model. Useful for placing particle effects and dynamic lights.

ROTATE Spawnflag (=32)

If set, the model_train will turn towards the next path_corner using its current **yaw_speed** and **pitch_speed** values (degrees/second).

Combining ROTATE and ROTATE_CONSTANT internally sets the **SPLINE** spawnflag and turns those flags off.

ROTATE_CONSTANT Spawnflag (=64)

If set, the model_train will constantly rotate around one or more axes continuously, until the train is toggled off or the rotational speeds are changed by a path_corner.

Combining ROTATE and ROTATE_CONSTANT internally sets the **SPLINE** spawnflag and turns those flags off.

SMOOTH_MOVE Spawnflag (=128)

If set, model_train will skip the last 0.1 second slowdown before reaching a path_corner so that it maintains a constant velocity. For more information see the description for **SMOOTH_MOVE** in the

func_train reference.

NOT_IN_EASY Spawnflag (=256)

The model_train will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The model_train will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The model_train will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The model_train will be inhibited and not appear when deathmatch=1.

SPLINE Spawnflag (=4096)

The model_train will follow a spline curve that is defined by the path_corner origins and angles vector.
For more information see the description of the func_train's **SPLINE** spawnflag.

[Lazarus Main Page](#)

Model_train

Model_train combines the features of model_spawn and func_train, allowing you to move md2 models around your map. Other than the obvious, there are at least two benefits to this capability: 1) You can create a window model, give it the RF_TRANSLUCENT renderfx and/or EF_SPHERETRANS effect, and team moving transparent windows with your func_doors, func_trains, etc. 2) You can use model_spawn's NO_MODEL spawnflag coupled with one of the rendering effects and simulate a moving spotlight. All settings are as described under "Model spawner", of course with the addition of func_train parameters. There are a few differences in the behavior of this entity from a func_train that you should be aware of:

- The model_train follows a series of path_corners just as a func_train does, but unlike a func_train the model_train is placed such that it's origin is at the path_corner, rather than it's minimum bounding box coordinates. You'll need to know where the origin of the md2 model is to be able to correctly place the model_train.
- The lighting of md2 models, unlike brush models, is dependent on the location of the model. Stated more correctly, the model is lit according to the location of its origin. This is generally a **good** thing, but can cause unexpected results. For example, if your window model has its origin at the center, and the door that the window is part of closes into a wall such that the origin of the window is hidden, the entire window will become dark as soon as the window origin passes into the wall.

Spawnflags:

TOGGLE (=2)

Start active, when triggered become inactive

PLAYER (=8)

Set this if you want to use a player model. If **usermodel** is blank, model_train will use the model of the player. Otherwise the model found at players/<usermodel>/tris.md2 will be used. Note that if you specify a player model using usermodel, it **must** be a model that has been modified as described in the **ActorPak** documentation, or the model will not be skinned.

NO_MODEL (=16)

Don't use a model. Useful for placing particle effects and dynamic lights.

Key/Value pairs

style

Specifies the animation type to use.

0: None (unless startframe and framenumbers are used)

1: ANIM01 - cycle between frames 0 and 1 at 2 hz

2: ANIM23 - cycle between frames 2 and 3 at 2 hz

3: ANIM_ALL - cycle through all frames at 2 hz

4: ANIM_ALLFAST - cycle through all frames at 10 hz

Note: The animation flags override **startframe** and **framenumbers** settings you may have entered. ANIM_ALL and ANIM_ALLFAST don't do what you might think - rather than setting a framerate, these apparently cause the model to cycle through its ENTIRE sequence of animations in 0.5 or 0.1 seconds... which for monster and player models is of course not possible. Looks exceptionally goofy.

usermodel

The model to load ("models/" will be tacked on to the front, or if **PLAYER** is set, "players/" will be tacked on to the front and "/tris.md2" to the end).

startframe

The starting frame : default 0

framenumbers

The number of frames you want to display after startframe

health

If non-zero **and** solidstate is set to SOLID_BBOX, the entity will be shootable. When destroyed, it blows up with a no-damage explosion.

solidstate

1. SOLID_NOT - not solid at all. These models do not obey any sort of physics. If you place them up in the air or embedded in a wall they will stay there and be perfectly happy about it.
2. SOLID_BBOX - solid. These models will "droptofloor" when spawned. If the health value is set they may be damaged.

NOTE : if you want the model to be solid then you must enter vector values into the following fields:

bleft

the point that is at the bottom left of the models bounding box in a model editor

tright

the point that is at the top left of the models bounding box in a model editor

effects

1: ROTATE Rotate like a weapon
2: GIB
8: BLASTER Yellowish orange glow plus particles
16: ROCKET Rocket trail
32: GRENADE Grenade trail
64: HYPERBLASTER BLASTER w/o the particles
128: BFG Big green ball
256: COLOR_SHELL
512: POWERSCREEN Green power shield
16384: FLIES Ewww
32768: QUAD Blue shell
65536: PENT Red shell
131072: TELEPORTER Teleporter particles
262144: FLAG1 Red glow
524288: FLAG2 Blue glow
1048576: IONRIPPER
2097152: GREENGIB
4194304: BLUE_HB Blue hyperblaster glow
8388608: SPINNING_LIGHTS Red spinning lights
16777216: PLASMA
33554432: TRAP
67108864: TRACKER
134217728: DOUBLE Yellow shell
268435456: SPHERETRANS Transparent
536870912: TAGTRAIL
1073741824: HALF_DAMAGE
2147483648: TRACKER_TRAIL

renderfx

1: MINLIGHT Never completely dark
2: VIEWERMODEL
4: WEAPONMODEL
8: FULLBRIGHT
16: DEPTH HACK
32: TRANSLUCENT Transparent
64: FRAME LERP
128: BEAM
512: GLOW Pulsating glow of normal Q2 pickup items
1024: SHELL_RED
2048: SHELL_GREEN
4096: SHELL_BLUE
32768: IR_VISIBLE
65536: SHELL_DOUBLE
131072: SHELL_HALF_DAMAGE White shell
262144: USE_DISGUISE

movewith

Targetname of the entity to move with



Model_turret

The **Lazarus** model_turret has all of the functionality of and is identical in operation to a **turret_breach**, but uses an .MD2 model rather than a brush model. See the **turret2** example map for a couple of examples using camera and machinegun models from the **Deadnode 2** mod.

In general, model_turret is best used as a **TRACK** security camera or automated turret, rather than a player-controllable or standard turret_driver-operated turret. Although you **can** build a standard turret, bounding box problems may prove to be too much of a problem to make this practical. The bounding box defined by **bleft** and **tright** does **not** rotate, so for turrets that are much longer than wide, either parts of the turret will be non-solid or the bounding box will hurt the driver as the turret rotates. Alternatively, you can build a normal turret_breach using one or more clip brushes, and team a nonsolid **model_spawn** gun/camera model with the turret_breach. Nonsolid teammates of the Lazarus turret_breach will both yaw **and pitch** with the turret.

Shown below are key value pairs unique to model_turret. All other parameters are as described in the **turret_breach** documentation.

Key/value pairs

bleft

The point that is at the bottom left of the model's bounding box in a model editor.

tright

The point that is at the top left of the model's bounding box in a model editor.

usermodel

The model to load ("models/" will be tacked on to the front).

[Lazarus Main Page](#)

Monster_*

Lazarus adds many new options to monster_* point entities, such as customizing monster hit points (**health**), creating monster teams (**dmgteam**), and adding support for the use of custom skins (**style**), gibs (**gib_type**) and blood color (**blood_type**). **Lazarus** also enhances their AI in more than a few ways. These AI enhancements can be seen in all standard single-player Quake2 maps, not just those which use the new **Lazarus** keys and spawnflags. The entities affected are:

- monster_berserk
- monster_boss2
- monster_brain
- monster_chick
- monster_flipper
- monster_floater
- monster_flyer
- monster_gladiator
- monster_gunner
- monster_hover
- monster_infantry
- monster_jorg
- monster_medic
- monster_mutant
- monster_parasite
- monster_soldier
- monster_soldier_light
- monster_soldier_ss
- monster_supertank
- monster_tank
- monster_tank_commander

Explanations of what the new features are can be found in the individual keyvalue and spawnflag descriptions below. There are many AI enhancements, too - for that info please see [this page](#).

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the facing angle of the monster on the XY plane. Default=0.

angles

Specifies the facing angle of the monster in 3 dimensions as defined by pitch, yaw, and roll. Default=0 0 0.

blood_type

Specifies alternate blood particles. Currently only 1 new type is supported: if blood_type=1, the monster will bleed gekk-yellow blood and his gibs will leave a gekk-yellow particle trail. Default=0.

combattarget

Targetname of the point_combat the monster will move to when angered.

deathtarget

Targetname of the entity to be triggered upon the monster's death.

distance

Specifies the maximum distance in map units that the monster can be from the player and will still see and shoot at him. Minimum value is 500 (values<500 are reset to 500 by the code). Default=1280.

dmgteam

An monster with a dmgtteam value set will treat all attacks upon other actor(s) or monster(s) with the same dmgtteam value as an attack on himself. Also see [this page](#). Dmgteam may also be used to trigger a [trigger_relay](#).

flies

Specifies the probability that the monster's corpse will produce flies. When they appear, they will have a duration of 60 seconds before they expire. Flies will never appear if the monster's corpse is in a liquid. Normal probability values range from 0.00-1.00; 0.00=0% probability (never produce flies); 1.00=100% probability (always produce flies). If flies>1, the monster will have flies buzzing about him even while alive. Default is dependent on monster class; see [defaults table](#). Ignored if **health**<0, unless flies=1. In that event, the initially dead monster will have flies which never expire. More about flies [here](#).

gib_health

Specifies the number of hit points required to gib the monster's corpse. Ignored when **NO_GIB** is set. Default is dependent on monster class; see [defaults table](#).

gib_type

Specifies alternate gib models should be used. If non-zero, the code replaces "gibs" in the gib model name with "gib1", "gib2", etc. Models are **NOT** supplied with Lazarus. Depending on monster type, you'll need up to 3 new models for each new type. Default=0.

health

Specifies the number of hit points required to kill the monster. For the monster_jorg, this is the number of hit points required to "kill" him and spawn the Makron.

If health=0 it will be reset to the default.

If health<0 the monster will be dead on startup.

Default is dependent on monster class; see [defaults table](#).

Initially dead monsters do not appear in the monster count nor in the kill count.

health2

Only applicable to the monster_jorg. Specifies the number of hit points required to kill the Makron, after the large version has already been "killed".

If health=0 it will be reset to the default.

If health<0 the Makron will be dead on startup.

Default=3000.

item

Classname of the entity to be spawned by the monster upon his death. Unlike the standard game, Lazarus monsters may drop any health item on death.

killtarget

Targetname of the entity to be removed from the map upon the monster's death.

movewith

Targetname of the parent entity the monster is to [movewith](#).

mass

Weight of the monster. Default is dependent on monster class; see [defaults table](#).

mass2

Only applicable to the monster_jorg. Weight of the Makron, after the large version has already been "killed". Default=500.

powerarmor

Specifies amount of optional power shield armor the monster has. Default=0.

This key is ignored for the monster_brain, who has his own power screen by default; also ignored for the monster_flipper and monster_mutant, which wear no clothes at all, much less a power shield.

Experiment with settings using the weapons the monster will face - it looks odd for a monster to die before his power shield runs out.

style

Specifies the 0-based index of the skin the monster will use. Please also read [these notes](#). Choices are:

0: standard skin (default)

- 1: 1st custom skin
- 2: 2nd custom skin
- 3: 3rd custom skin

target

Targetname of the path_corner the monster will move to.

targetname

Name of the specific monster.

Spawnflags

AMBUSH Spawnflag (=1)

Identical to **SIGHT**, except that the code will not call any idle sounds.

TRIGGER_SPAWN Spawnflag (=2)

The monster won't appear in the map until his targetname is called. Also, the monster count will not reflect that monster until he is spawned; at that time the monster count is updated.

SIGHT Spawnflag (=4)

The monster won't attack until angered or he has line-of-sight to his enemy. This is the same as normal Quake2's AMBUSH flag.

GOOD_GUY Spawnflag (=8)

The monster will not attack the player, and will treat all attacks upon the player as an attack on himself. If the player shoots a GOOD_GUY monster, the GOOD_GUY flag is removed and the monster will then turn on the player.

NO_GIB Spawnflag (=16)

The monster cannot be gibbed. Visual evidence is that the corpse produces sparks rather than blood when shot.

SPECIAL Spawnflag (=32)

The monster uses an attack other than the default. As of the current **Lazarus** version, this affects only rocket-firing monsters - they will use homing rockets instead of standard rockets. Homing rockets are somewhat slower than normal rockets, which actually makes them deadlier since they have more time to correct themselves. This flag is ignored for monsters without a SPECIAL attack option.

IGNORE_FIRE Spawnflag (=128)

The monster can shoot other monsters/actors without them getting mad at him. This is useful flag to set for a monster whose placement frequently causes him to hit other monsters/actors with friendly fire. Setting this flag does **not** mean that IGNORE_FIRE monsters and **GOOD_GUY** monsters/**actors** won't shoot each other.

NOT_IN_EASY Spawnflag (=256)

The monster will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The monster will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The monster will be inhibited and not appear when skill=2 or greater.

Defaults Table

Many of the above keys use default values that are monsterclass-dependent.

--	--	--	--	--

	flies	gib_health	health	mass
monster_berserk	0.20	-60	240	250
monster_boss2	N/A	N/A	2000	N/A
monster_brain	0.10	-150	300	400
monster_chick	0.40	-70	175	200
monster_flipper	0.90	-30	50	100
monster_floater	N/A	N/A	200	N/A
monster_flyer	N/A	N/A	50	N/A
monster_gladiator	0.05	-175	400	400
monster_gunner	0.30	-70	175	200
monster_hover	N/A	N/A	240	N/A
monster_infantry	0.40	-40	100	200
monster_jorg	0.00	-900	3000/3000	1000/500
monster_medic	0.15	-130	300	400
monster_mutant	0.90	-120	300	300
monster_parasite	0.35	-50	175	250
monster_soldier	0.40	-30	30	100
monster_soldier_light	0.40	-30	20	100
monster_soldier_ss	0.40	-30	40	100
monster_supertank	N/A	N/A	1500	800
monster_tank	0.05	-200	750	500
monster_tank_commander	0.05	-225	1000	500

Custom skins

Quake2 models reference skins from within the model itself. If that model does not contain references to additional skins (and Q2 monster models don't), then additional skins cannot be used unless the model is modified. But since distributing what are essentially just copies of id Software's models is a no-no, this exercise is pointless.

Lazarus solves this problem by generating modified versions of necessary monster models, which **do** reference skins, on the fly. This also means that you won't have to distribute them along with your maps (nor should you, since that would violate id Software's EULA). If your map uses a monster which calls a custom skin, when the user runs your map, the new modified model will be generated on the user's machine. This means no copyright violations for you, and a smaller download for the user. (*Auto-generating model code courtesy of **Argh!***).

Custom skins for monsters are set with the **style** key, and the code won't unnecessarily generate a model if all monsters which use that model lack a style value of greater than 0. Each monster class can use up to 4 different skins (1 standard game skin and 3 custom skins. Support for pain versions of each skin is supported as well).

This means you can identify monsters, with different abilities (like **SPECIAL** iron maidens) or monsters which use **power armor**, with different skins. While **Lazarus** provides the ability to use custom monster skins, at this time it sadly does not include the custom skins themselves - that's up to you to provide.

Here's an example of how to use this feature: Place a monster (let's use an enforcer) in your map and set its **style**=1. Then place 2 new skins (custom skin and the pain skin) in /QUAKE2/[gamedir]/models/monsters/infantry/, and name them "custom1.pcx" and "custompain1.pcx". Compile the map and run it in the game, and that particular enforcer should be using the new skins.

The filenames and file locations you'll have to use for custom skins is for the most part a standardized affair; but there are a few exceptions. All is explained in the table below.

monster_soldier_light This monster shares its model with the monster_soldier, and its skins reside in the same place, and therefore use non-standard filenames.	- standard skin: models/monsters/soldier/skin_lt.pcx models/monsters/soldier/skin_ltp.pcx - 1st custom skin: models/monsters/soldier/custom1_lt.pcx models/monsters/soldier/custompain1_lt.pcx - 2nd custom skin: models/monsters/soldier/custom2_lt.pcx models/monsters/soldier/custompain2_lt.pcx - 3rd custom skin: models/monsters/soldier/custom3_lt.pcx models/monsters/soldier/custompain3_lt.pcx
monster_soldier_ss This monster shares its model with the monster_soldier, and its skins reside in the same place, and therefore use non-standard filenames.	- standard skin: models/monsters/soldier/skin_ss.pcx models/monsters/soldier/skin_ssp.pcx - 1st custom skin: models/monsters/soldier/custom1_ss.pcx models/monsters/soldier/custompain1_ss.pcx - 2nd custom skin: models/monsters/soldier/custom2_ss.pcx models/monsters/soldier/custompain2_ss.pcx - 3rd custom skin: models/monsters/soldier/custom3_ss.pcx models/monsters/soldier/custompain3_ss.pcx
monster_tank_commander While this monster shares its model with the monster_tank, its skins are in a different location. This "location" for the pakfile is given to the right; note "../" for pakfile purposes is a "directory". Pakfiles really don't have directories, but pakfile browsers display them that way. If you're testing a monster_tank_commander skin on disk, then use the actual disk location "models/monsters/ctank/".	- standard skin: models/monsters/tank/../../ctank/skin.pcx models/monsters/tank/../../ctank/pain.pcx - 1st custom skin: models/monsters/tank/../../ctank/custom1.pcx models/monsters/tank/../../ctank/custompain1.pcx - 2nd custom skin: models/monsters/tank/../../ctank/custom2.pcx models/monsters/tank/../../ctank/custompain2.pcx - 3rd custom skin: models/monsters/tank/../../ctank/custom3.pcx models/monsters/tank/../../ctank/custompain3.pcx
monster_[classname] This is representative of all types not listed above.	- standard skin: models/monsters/[monster]/skin.pcx models/monsters/[monster]/pain.pcx - 1st custom skin: models/monsters/[monster]/custom1.pcx models/monsters/[monster]/custompain1.pcx - 2nd custom skin: models/monsters/[monster]/custom2.pcx models/monsters/[monster]/custompain2.pcx - 3rd custom skin: models/monsters/[monster]/custom3.pcx models/monsters/[monster]/custompain3.pcx

Those Damned Flies

The effect behind the **flies** key is the same as you've become accustomed to when killing enforcers and mutants. **Lazarus** adds the flies effect to all non-flying monsters (even the fish, should you knock him out of the water). When no value is entered, the **default** probability values are used - this means you'll see other monsters produce flies in any

map played under **Lazarus**. Some monsters are more likely to draw flies than others. This is in direct proportion to the flesh content of the given monster, so you'll see mutants drawing flies consistently, while gladiators and tanks will seldom draw flies. Of course, for your own map, you can set the probability of the appearance of flies to whatever you want.

[Lazarus Main Page](#)

Monster_commander_body

The **Lazarus** monster_commander_body is identical to the standard Quake2 monster_commander_body point entity, with the addition of the ubiquitous **movewith** support, and the **FLIES** spawnflag. This last offers the option of creating the pleasant atmosphere of having flies buzzing around him, in the same manner as often seen with enforcer and mutant corpses.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies facing angle on the XY plane. Default=0.

angles

Specifies facing angle in 3 dimensions, defined as pitch, yaw, and roll. Default=0 0 0.

movewith

Targetname of the parent entity the commander's body is to **movewith**.

targetname

Name of the specific commander's body.

Spawnflags

FLIES Spawnflag (=128)

Commander's body has flies buzzing around him.

NOT_IN_EASY Spawnflag (=256)

Commander's body will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

Commander's body will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

Commander's body will be inhibited and not appear when skill=2 or greater.

Monsters

dmgteam - New key/value pair. When a monster is hurt, all monsters with the same dmgtteam value will attack the **player** that attacked their dmgtteam teammate, and/or the **monster** that attacked if the action is initiated by a **target_monsterbattle**. This key is particularly useful for surprise attacks on players who like to blaster-snipe at monsters from a "safe" position.

Do **not** use "player" as a dmgtteam name... this has a special meaning to "good guy" monsters (see below).

GOOD_GUY spawnflag (=8) - If set, monster becomes a player bodyguard. Good guys don't automatically attack normal monsters or vice versa. Instead, good guys attack anybody who hurts the player. If the player hurts a good guy, the good guy is allowed to defend himself, but his dmgtteam teammates don't come to his aid. **If any artists are paying attention here... this feature could really make use of custom skins.**

NO_GIB spawnflag (=16) - If set... you guessed it, the monster cannot be gibbed. This is for all of you really mean medic fans out there :-)

HOMING_ROCKET spawnflag (=32) - If set, boss2, chick, supertank, tank, and tank_commander will fire **homing rockets** at their targets. These things are nasty. Ideally I'd like to have new skins for homer-firing monsters, but that would require either also distributing the original models (modified to use new skins if this flag is set... and we're pretty sure that's illegal) or building completely new models... and to be honest, you guys don't pay us enough for that :-). So - personal opinion inserted here: **If you use** this spawnflag on a monster of a given class, then two things should happen: 1) **All** monsters of the same class should use homers to avoid confusion, and 2) You should make it very clear that homing rockets are used in the map, and we don't just mean in a text file, either. In our opinion this feature should be incorporated into the map's story somehow... for example have the help computer warn the player that he's coming upon a laboratory where the Strogg are experimenting with new weapons. End of speech :-) The homing rockets fire a bit slower than normal speed, which actually makes them deadlier since they have more time to correct themselves. Your opinion on homing rocket speed and path correction is welcome.

health, gib_health, and mass - You can now set these values for monsters on an individual basis. Setting the values to 0 will give the monster his default values. Setting the health to a negative number will result in the monster dying at map startup, so you can easily populate your map with dead monsters for a little added atmosphere :-). Setting gib_health means using negative health values. Mass will determine how much a monster is knocked back by weapon fire and explosions. **Special case:** The Makron's health and mass values are controlled by the new **health2** and **mass2** values for the Jorg.

powerarmor - You can optionally give monsters power shields, complete with the green "flash" effect when they're shot. This does not apply to the monster_brain, who has his own power screen by default; nor is it available to the monster_flipper and monster_mutant, which wear no clothes at all, much less a power shield. Experiment with settings using the weapons the player would have at the time they encountered shielded monsters in your map - it looks odd for a monster to die before his power shield runs out.

movewith - This is set to the targetname of the parent entity you wish the monster to **movewith**. The monster will be locked in place relative to its parent, but this can have advantages in some cases.

monster tallies - Trigger spawned monsters are no longer included in the monster totals until they spawn into the game. Initially dead monsters (created by setting health < 0) are never counted in the monster totals. Killtargeted monsters are removed from the total monster count.

Monster AI

Gunners

Gunners are much better grenadiers than in any previous version of Q2. They will always check to ensure that a grenade can hit their target, and choose the machinegun if not. If a grenade **can** hit the target, they use the correct elevation angle to get the grenade there... this change makes the gunner a much more formidable opponent. Features:

- If the gunner is at the same elevation as or above the target, he'll aim at the target's feet to maximize the probability of splash damage.

If the gunner is below the target, he'll aim at the center of the target to help minimize the risk of the grenade's randomness causing it to hit the platform the target is standing on.

- This is cool... gunner will sometimes lead a moving target. His tendency to lead is relative to skill level - on easy he'll lead 20% of the time; on nightmare 65% of the time. The randomness in this decision makes it impossible to predict what the gunner will do.
- If the gunner is at the same elevation as or up to 160 units above the target, he checks to make sure that his "perfect" trajectory would not hit a solid object before reaching the target. In other words... if a gunner is on one side of an open doorway with the player on the other side and the calculated trajectory would cause the grenade to hit above the door, the gunner adjusts his aim down to get the grenade through the door, hoping for a good bounce.
- The current sv_gravity setting is taken into account, both when checking to see whether grenades can reach the target and when calculating a good trajectory.

Medic

- The medic's cable now originates at the correct location. You've probably noticed in standard Q2 that the start point of the cable is fouled up if the medic is close to the dead monster he's attempting to heal.
- Medic will no longer attempt to heal a dead monster if that operation would result in the monster and medic bounding boxes intersecting... which makes a bit of a mess.
- Medic will no longer attack another monster, even if the other monster deliberately attacks the medic. This avoids a bit of Q2 silliness - medic killing a monster then immediately healing him.

Tanks/Chicks

Several changes, most from Rogue's Mission Pack:

- Monster will no longer fire suicidal rocket blasts into adjacent walls when the player strafes out of view.
- Monster will lead targets when firing rockets. As with the gunner, the tendency to lead the target is tied to skill level: Easy=20%, Normal=35%, Hard=50%, Nightmare=65%. This does not apply if homing rockets are used.
- If the target's feet (or minimum z bounding box coordinate) is below the launcher, the monster will fire at the target's feet 2/3 of the time. In all other cases the monster will fire at the target origin.
- Rocket speed is skill level dependent ($500 + 100 * \text{skill}$). That's a rather speedy 800 units/sec on nightmare. Again, this change does not apply to homing rockets.

Bug fixes

The original version of Quake2 contains several annoying AI bugs that are fixed by Lazarus:

- In standard Q2, if a monster is patrolling between **path_corners** and a path_corner has a wait value >0, when the wait expires it will start walking in the same direction, not towards the next path_corner. The problem is solved by Lazarus.
- Also in standard Q2, if a monster's combattarget point_combat is oriented at 180 degrees from the monster, when it becomes angry the monster will take off at 0 degrees and run until it is obstructed by another entity, then realize its mistake and turn around. This too has been fixed.

Self-preservation

We don't like to watch monsters suffer needlessly (we want to **make** them suffer), so Lazarus gives monsters a temporary entity to get mad at and chase to a safer area when they are damaged by environmental effects. For example, in standard Quake2 if a monster happens to be standing in a rather bad spot when a func_water lava brush rises up to fry him, it will continue standing in the same spot, contemplating what to have for dinner. In Lazarus this same situation will result in the monster attempting to exit the lava.

Movewith key/value pair

The movewith key/value pair opens up a realm of possibilities for entity tricks that previously required complicated scripted sequences or were just plain not possible. This functionality has been a part of **Lazarus** from the beginning with the **func_trainbutton**, but since v1.2 we've been seriously continuing to expand and improve upon this feature.

So enough hype already, what's this all about? The movewith key/value pair allows you to tie the motion of just about any entity to the motion of (currently) a **func_train**, **func_tracktrain**, **model_train**, **func_door** or a **func_vehicle**. These parent/child assignments are typically done on map load, however it's possible to form new movewith associations and terminate existing associations **while the map is running** by using a **target_movewith**.

What's this thing good for? Here are a couple of examples:



That's a **func_train** landing craft that has moved up to the beach, which pitched upwards to land on shore, and then lowered its **func_door_rotating** door to deliver **monster_soldiers** on the scene. In the background is a **func_train** helicopter with **func_rotating** rotors and a crew of monsters. The map shown above is available in the **example maps**. (The helicopter is a modified version of the prefab created by Hallucinator, available at the **Rust Prefabs**. The palm tree was created by Monsto Brukes for the Jump Beach map, also available from Rust if you look hard enough).

A few other examples of new possibilities:

- Would you like a tank to travel along a predefined path and fire rockets at players? Use a **turret_breach** with the **TRACK** spawnflag and set it to *movewith* the **func_train** chassis.
- Doors on vehicles? No problem :-). Note that you still cannot have moving transparencies - that's a whole different problem.
- Popup turrets can now be built very easily, without resorting to the earlier trick of building a duplicate **func_train** which was **killtargeted** and its place taken by a **TRIGGER_SPAWN turret_breach**. And an added bonus here is that the turret can fire while moving into position.
- Want a **func_train** to emit particle effects? Couple a **target_effect** with a **func_train**. You might also want to set the **IF_MOVING** spawnflag on the **target_effect** so that the effect will only be used when the **func_train** is in motion.

Who can use *movewith*? Good question. To date we've tested *movewith* with these entities successfully, though not in all possible configurations:

- ammo_*
- func_button
- func_door
- func_door_rotating
- func_explosive
- func_rotating
- func_wall
- info_notnull
- item_*
- key_*
- misc_actor
- misc_deadsoldier
- misc_insane
- misc_teleporter
- misc_teleporter_dest
- model_spawn
- monster_*
- monster_commander_body
- target_attractor
- target_blaster
- target_effect
- target_explosion
- target_laser
- target_playback
- target_spawner
- target_speaker
- target_splash
- target_temp_entity
- tremor_trigger_multiple
- trigger_bbox
- trigger_hurt
- trigger_inside
- trigger_look
- trigger_mass
- trigger_monsterjump
- trigger_multiple
- trigger_once
- trigger_push
- trigger_teleporter
- turret_breach
- turret_base
- weapon_*

And, as mentioned previously, the target of movewith must currently be a func_train, model_train, func_door, or func_vehicle.

Look, But Don't Touch

Simple movewith parent/child setups work pretty well, like buttons moving with trains, but more complicated combinations (especially involving rotating bmodels) can be pretty fragile. You can work out all the math and get everything running in exact and perfect sync, and yet everything can go to hell if any part of the movewith parent/child combo gets blocked by a player or monster. Even being blocked by a dropped item can cause problems sometimes.

There's no way to foresee every possible movewith construction that could possibly be conceived, but we're going to try and offer some tips anyhow:

- **Protect the children.** Sometimes blocking the parent causes no problems, but blocking the child does. One possible solution is to add clip brushes to the parent bmodel, so that the child is completely encased by the parent.

- **You can't block it if it isn't solid.** If both parent and child are made from mist brushes, they should not be blocked by anything. Of course non-solid moving brush models may not be what you had in mind.
- **Lock it away safe and sound.** Encase the whole construction inside of clip brushes, or put in an inaccessible area so that the player can see it through transparent brushes or through gaps he can't fit through. In other words, make it into pure eye-candy.

Making movewith parent/child combos act as if they are also teamed (block one and they're both blocked) is sort of a holy grail for us. We're working on improving this, but we're beginning to suspect that we'll probably never be satisfied no matter how much we improve it.

Parent/Child Heirarchy and Entity Map Order

Now that the func_door has been added as a possible movewith parent, it's pretty easy to look forward to making some cool stuff where you could have a func_door act as both a child and a parent at the same time. Uh, no... that doesn't work out too well. The problem is that the child of this func_door child-as-parent really doesn't know when to move and when to stop relative to the parent's parent. (Did any of that make sense?) This is a timing issue that will likely improve. Whether it improves enough to be usable remains to be seen - please keep in mind that the Quake2 engine was never really designed to do this stuff.

This issue of timing comes up again where map order is concerned. When we speak of map order, we're referring to the order that different entities appear in the .map (and therefore the compiled .bsp) file. If you've done any experimenting with constructing movewith parent/child combos, you may have run into instances where the child always seems to lead the parent around. When this happens, you're seeing the effect of entity map order. The solution is to ensure that **the parent occurs in the map later than the child**. How you do this is map editor-specific... you could edit the .map file directly if you feel you must, but we don't recommend it (unless you're a Radiant user). We will tell you that it can be done in Worldcraft very easily: Select the parent bmodel, right-click and delete it (now it's gone). On the Edit menu, click 'Undo'. Now it's back, with all of its keys and flags intact - and most importantly, as a new entity written to the end of the map file.

Rotating Train Considerations

For the most part (with exceptions noted below), all of the entities mentioned above will movewith a rotating train with no serious problems. However, you will need to decide early on how you expect those entities to move as the train rotates. Should the train rotation be applied to the movewith child, or should the child's angles be independent of train rotation? For example, the func_door_rotating of the landing craft shown above should both move and rotate with the landing craft func_train. Otherwise you'd have either a gap or an overlap between door and train as the train pitches up or down. On the other hand, for many entities you will want the child to translate with the train, but rotate independently. You can control this behavior with the func_train's **turn_rider** key. If non-zero, train rotations will be applied on top of whatever rotations the child is going through. If zero, train rotation is **not** applied to the child (though translation of the origin due to that rotation will still be performed).

Limitations

Of course there are several limitations as far as what will work and what will not, as well as a few physics problems you should be aware of before using *movewith*. Some of these problems will be improved upon or removed in the future, while some are fundamental limitations we can do nothing about.

- **Lights.** You cannot use *movewith* with a normal light, whether it is a named switched light or not. Lighting is calculated during the rad process, and we'll never be able to change that without buying the game engine. On the other hand, if you'd like a moving spotlight, then you can couple a TE_FLASHLIGHT **target_effect** with a func_train and it will work just fine (with the exception that brush models, including the coupled func_train, will be lit as though in their initial map position... this is a limitation of **all** lighting effects, not just the target_effect).
- **Func_door_rotating.** Rotating doors may be moved **as long as** the axis of rotation does not change orientation. For example, in the landing craft example above, the landing craft changes pitch before the func_door_rotating opens. This is fine since the door will still rotate around an axis with the same orientation as the door was designed

for. But if the landing craft yawed or rolled before opening the door, it would not work as expected. The landing craft **can** yaw or roll before the door opens, as long as the orientation of the train's axis at the time the door opens is correct for the rotating door. This limitation **may** be improved upon in future releases. Another limitation of rotating doors is that the func_train mover **must** be stationary as the door moves. The code doesn't attempt to enforce this limitation, but the results will be very strange if you violate it. Also... see notes on door triggers under [func_door notes](#) below.

- **Func_rotating.** Rotating brushes will maintain the correct orientation if the func_train goes through yaw changes, but as of yet we haven't been able to maintain the correct orientation for train pitch and roll changes. For that reason, the code currently doesn't even attempt to correct func_rotating angles in response to roll and pitch changes. So, for example, if the helicopter in the above picture rolls or pitches, the main rotor will remain in the horizontal plane. For small changes in pitch or roll this limitation isn't a big deal, but for large changes will look exceptionally strange (as well as kill everyone on board the chopper).
- **Func_door and func_button.** If used with non-rotating trains, these entities work without change. If used with rotating trains you'll need to jump through a few hoops to get things working correctly. First, the door or button will need an origin brush (which sadly means that texture alignment for these entities will be a problem). But just having an origin brush isn't enough - you want the door or button to rotate around the same axis that the train does. So, for example, with a train that goes through yaw rotations only, the origin brush for a func_door should have the same x and y coordinates as the train origin. If the train rotates in more than 1 direction then the origin for a door should match the train origin exactly. If you do **not** match up the origins in this way, then the door or button movement will be a bit sloppy as the train turns. It will catch up to the correct location every 0.1 seconds, but in between those times appear to drift out of place.

The second problem applies to doors only - how to trigger it. If you use a trigger_multiple to open a door, the trigger should also *movewith* the train, and if the train rotates the trigger must also use an origin brush. Here's where the fun starts: as the train rotates, **Lazarus** correctly rotates the trigger field for the door. However, Quake2 doesn't care about the actual trigger field shape or orientation - it only cares about the min/max coordinates of the trigger field. So at a yaw angle of 45 degrees or so, the trigger field bounding box will be much larger than it is with a yaw angle of 0, 90, 180, and 270 degrees. It's very possible that a player will be outside the trigger field at yaw=0, but inside the trigger field at yaw=45 even though the player never moves. Our suggestion: use another method to open the door - a func_button will work nicely here.

Lastly, both doors and buttons have a fairly sloppy motion if they are activated during a train turn. The problem isn't as noticeable with typical buttons since they are generally much smaller than doors. You'll likely have to play around with this a bit to get it to your liking. For best results you'll likely want to use a door that is much thinner than the wall it moves into.

- **Pickup items (and other point entities which use visible models).** Pickup items work fine, other than a bit of sloppiness during turns. The farther the item is from the train origin, the worse the problem will be. There's really no way around this other than to place the pickup item at the train origin, or of course to forego using pickups with rotating trains.
- **Speakers.** You **can** tie target_speakers to a func_train with *movewith*, but the results may not be what you intended. Looped sounds will work just fine, and move along with the func_train. However, as we've all come to know and despise, looped sounds attenuate much too rapidly to be useful in many situations. So... OK, you say, instead of a looped sound, use a normal non-looped target_speaker targeted by a func_timer and you can play it as loud as you want. The trouble here is that although the target_speaker origin moves with the train, the origin of the sound will **not** move once the sound starts playing. What it will do is play the sound from start to finish at the position the speaker was when it was triggered. When triggered again, the sound will play at the target_speaker's new position.
- **Turret_breach and turret_base.** There is a lot of code behind these entities that assumes the turret_base will rotate around the z axis, and that the breach will pitch up/down and/or rotate with the turret_base around z. Things will fall apart rapidly if you use them with a func_train that goes through pitch or roll movements.
- **Monsters.** The problem with monsters isn't *movewith*... rather *movewith* is the solution. Monsters without the *movewith* value will invariably block the movement of a func_train that goes through pitch or roll changes, and end up being killed by the train. You can prevent this problem by using *movewith* with the monsters. The problem here of course is that the monsters will be stuck with the func_train - they won't be able to dismount. This situation will almost certainly improve. Players have a similar problem with pitching/rolling func_trains, but of course you cannot

use *movewith* with a player - but we're working on the problem.

Lazarus Main Page

Path_corner

The **Lazarus** path_corner has many capabilities beyond that of the standard Quake2 path_corner point entity, with changes that fall into 4 categories. The first is the addition of **count** support, which allows it to be auto-killtargeted. The second concerns changes to support **Lazarus** rotating **func_trains**. The keys/flags which do this are **roll**, **speed**, **pitch_speed**, **yaw_speed**, and **roll_speed**; and **NO_ROTATE**. The third has to do with **target_locator** support - **angle/angles** values set by the path_corner are inherited by the entity the target_locator will spawn there. And finally, the Lazarus path_corner can **movewith** a func_train so that the motion of the train targeting **this** path_corner is an ever-changing path.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Relevant in two cases:

- The path_corner is referenced by a **target_locator**. If non-zero, determines the facing angle on the XY plane of the entity which will spawn at the path_corner.
- The path_corner is referenced by a **func_train** or other moving entity that follows a **spline** curve. In this case, the angle is used to define the shape of the curve. If you want the path to curve in a vertical plane as well as horizontal, **OR** if you want the train to roll between path_corners, use the **angles** vector instead of angle.

Default=0.

angles

Relevant in two cases:

- The path_corner is referenced by a **target_locator**. If non-zero, determines the facing angle in 3 dimensions (as defined by pitch, yaw, and roll) of the entity which will spawn at the path_corner.
- The path_corner is referenced by a **func_train** or other moving entity that follows a **spline** curve. In this case, the pitch and yaw angles are used to define the shape of the curve, and the train's roll angle with change linearly between the start and end path_corner's roll angle (the third component of the angles vector). Positive roll values will cause the train to rotate toward the center of a turn.

Default=0 0 0.

count

When non-zero, specifies the number of times the path_corner will be used as a path node before being auto-killtargeted (see [this page](#) for details). Default=0.

movewith

Targetname of the parent entity the path_corner is to **movewith**.

pathtarget

Targetname of the entity to be triggered when the path_corner is used.

pitch_speed

Specifies the absolute rotational speed on the pitch axis of a **ROTATE func_train** which uses the path_corner; and specifies the change in rotational speed on the pitch axis of a **ROTATE_CONSTANT func_train** which uses the path_corner. Default=0.

roll

Specifies the roll angle that a **ROTATE func_train** should turn to while moving towards this path_corner. The func_train's **roll_speed** must be non-zero to perform this rotation. Default=0.

roll_speed

Specifies the absolute rotational speed on the roll axis of a **ROTATE func_train** which uses the path_corner; and specifies the change in rotational speed on the roll axis of a **ROTATE_CONSTANT func_train** which uses the path_corner. Default=0.

speed

Specifies the change in speed of a func_train which uses the path_corner. Default=0.

target

Targetname of the next path_corner in the path.

targetname

Name of the specific path_corner.

yaw_speed

Specifies the absolute rotational speed on the yaw axis of a **ROTATE func_train** which uses the path_corner; and specifies the change in rotational speed on the yaw axis of a **ROTATE_CONSTANT func_train** which uses the path_corner. Default=0.

wait

Specifies the amount of time in seconds that an entity which uses the path_corner will pause before proceeding to the next path_corner in the sequence. If wait=(-1), the entity will not proceed unless it is triggered to move again. Default=0.

Spawnflags

TELEPORT Spawnflag (=1)

An entity set to move to this path_corner will appear there immediately rather than moving there in its normal manner.

NO_ROTATE Spawnflag (=2)

A **ROTATE** func_train will not change its orientation to turn towards this path_corner.

NOT_IN_EASY Spawnflag (=256)

The path_corner will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The path_corner will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The path_corner will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The path_corner will be inhibited and not appear when deathmatch=1.

Path_track

Path_track is to **func_tracktrain** as **path_corner** is to **func_train**: it is used to define a path that the train will follow.

Key/value pairs

deathtarget

Targetname of the entity to trigger when the func_tracktrain reaches this path_track, and this path_track is a dead end. Useful for automatic **func_trackchange** operation.

pathtarget

Targetname of the entity to trigger each time a func_tracktrain reaches this path_track.

speed

When a func_tracktrain passes through this path_track, its maximum speed is multiplied by this value. Note that this is a **ratio**, not an absolute speed... so don't go setting it to 500 and say we didn't warn you :-)

target

Targetname of the next path_track on the func_tracktrain's path. See notes below concerning **train route**.

target2

Targetname of an alternate path_track for the func_tracktrain's path. For normal one-way paths, this value will only be used with **ALTPATH** path_tracks. For two-way paths this value will be required for at least one path_track that lies on the main line between 2 ALTPATH branch points. Again, see **train route** below.

targetname

Name of the path_track. This field is required.

Spawnflags

ALTPATH Spawnflag (=1)

If set, this path_track is a branch point. Target and target2 specify alternate path_tracks to travel to from this path_track. Track and track2 are swapped each time this path_track is triggered, and the train will always travel to "target" (assuming the map is constructed properly and "target" passes the tests detailed below under **train route**).

DISABLED Spawnflag (=2)

A train may not travel to this path_track. If **ALTPATH** is **not** set, DISABLED is toggled each time a path_track is triggered.

FIREONCE Spawnflag (=4)

If set, the path_track's **pathtarget** will only be triggered one time.

DISABLE_TRAIN Spawnflag (=8)

When reaching this path_track the train will set its **NOCONTROL** spawnflag. If a player is currently in control of the train, he will lose that control.

NOT_IN_EASY Spawnflag (=256)

The func_train will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The func_train will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The func_train will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The func_train will be inhibited and not appear when deathmatch=1.

Train route

For simple HL-like train paths, routing is very simple. Func_tracktrain travels to its target path_track, then to its target's target, then to its target's target's target, then to... well, you know. Even when throwing **ALTPATH** path_track's into the mix the picture is still simple, since the train still travels to the path_track's target. Now add the ability to not just back up, but turn around and go the other way, and things get just a bit more complicated. Fortunately for the mapper, all the tedium of picking out a good next target is performed by the code, rather than forcing you to jump through a lot of hoops. The main feature that makes all this go is that the code will **never** allow the selection of a next path_track that would result in turning the train more than 90 degrees. This should not be a hindrance to you, as a 90 degree turn is much too severe for normal train operation. From the top, here are the procedures used to select the next path_track that the train will travel to:

If the train reverses direction:

- Does the currently targeted path_track have a "prevpath" member set? If so (and this should always be true) check the angle to that path_track. If less than 90 degrees, we're done. (This test should always pass, since prevpath is set every time a new target path_track is selected. We're just being cautious here). If the currently targeted path_track does not have a "prevpath" or if the prevpath path_track is at a bad angle, proceed below. Otherwise we're done.

If the train is moving forward:

- Does the current path_track have a "target"? If so, check the angle to this path_track. If more than 90 degrees, reject it.
- If the current path_track does **not** have **ALTPATH** set, and it has a "target2", check the angle to that path_track. If more than 90 degrees, reject it. If both "target" and "target2" are valid, select the one that is closest to the current forward direction.
- If we do not now have a valid target to travel to, check all path_tracks (other than the current one) for "target" or "target2" pointing to the current path_track. Apply the same angle checks, and if both are valid select the one that is closest to the current forward direction.
- If we still don't have a valid target, we must have reached a dead end. Stop and fire the **deathtarget**, if any. If we **do** have a valid target, then set that target's "prevpath" member to the current path_track.

If the train is moving backwards:

- If the current path_track has **ALTPATH** set, check the angle to the "target". If greater than 90 degrees reject it.
- If we don't have a valid target, and the current path_track has a "prevpath" path_track (meaning we've travelled forward through this path_track before), check "prevpath". If the angle to "prevpath" is greater than 90 degrees, reject it.
- If we don't have a valid target, check all path_tracks (other than the current one) for "target" or "target2" pointing to the current path_track. Apply the same angle checks, and if both are valid select the one that is closest to the current backwards direction.
- Finally, if we don't have a valid target then check the current path_track's "target" and "target2". If both are valid, select the one that is closest to the current backwards direction.
- If we still don't have a valid target, we must have reached a dead end. Stop and fire the **deathtarget**, if any.

In short, you should always be able to get from A to B if A->target=B, A->target2=B and A isn't ALTPATH, B->target=A, or B->target2=A. One particular scenario may cause you to grind your teeth a bit before you realize why things don't work correctly: You cannot have 2 adjacent ALTPATH path_tracks whose branches all point away from the other ALTPATH path_track. To overcome this, add a path_track between those 2 ALTPATH path_tracks that has "target" set to one of the junctions and "target2" set to the other.

Lazarus Main Page

Physics

As **Lazarus** has evolved, it has incorporated attempts to more accurately model physics within the Quake2 game world. This is so the interaction of entities in regards to their velocity and mass tend to act more logically and more like you'd expect that they'd act.

In many cases these changes are better documented on the pages that describe the entities themselves. As some examples, the descriptions of the actions and behavior of the `func_pushable`, `func_vehicle`, and `func_explosive` are best found on their respective pages. However there are some other changes that really aren't documented elsewhere, so this is what this page is about.

Velocity of moving brush models as they affect the player

You may have noticed that in the unmodified game, if the player is riding a horizontally-moving platform, and jumps straight up, the platform will continue on without the player. So the player will fall back down and not land on the platform again. This is not what one would expect to happen in the real world. An person riding on a moving object will have the same velocity as the object he's riding on. **Lazarus** changes this effect in the game so that the velocity component of a moving brush model the player may be riding on will be added to the player's velocity. In the case of the horizontally-moving platform, jumping straight up will mean the player will continue moving in that horizontal direction (since he's inherited the velocity of the platform) and will land on the platform again.

This effect extends to brush models moving up and down as well. If the player jumps just before a very-rapidly rising lift he's riding on reaches the top of its travel, the player will inherit the upward velocity of the lift and it will be added to his jumping velocity. Which means he may jump pretty high. A rocket jump at this moment should allow the player to achieve some pretty serious height :-)

Velocity of the player as it affects grenades

Grenades, both those that are thrown by hand and those launched from the grenade launcher, now have the player's velocity added to their own. This has proven to be a pretty significant change in the gameplay as far as grenades go. Where in the original game, the key to getting more range out of a grenade was to throw/fire at the top of a jump (greater height), in a **Lazarus** game this slight advantage that a small increase in height will give you is overshadowed by the advantage that an increase in velocity will give you. Try firing a grenade while running forward. Better, try firing a grenade while running forward and jumping at the same time (fire right after jumping, not at the top of the jump; remember, it's velocity you're after here). You'll fire grenades higher and further than you ever could before.

There's a flip side to this of course. Throwing/firing a grenade while running backwards (worse, if while falling) will result in a firing range that is severely shortened. Also (and this is cool), the player's strafing velocity is added to the grenade as well. Here's a tip: If you're hiding behind a corner, and you strafe out to throw a grenade, make sure that grenade is away before you strafe back behind cover. If you strafe back as quickly and as soon as you've become accustomed to in the original game, the grenade will move sideways too, and it will likely hit the object you're trying to hide behind and bounce back in your face.

Velocity of the player as it affects rockets

As an experiment, we're also adding the ability to add player velocity to a fired rocket in much the same way as it is for grenades. (This feature is **disabled** by default - if you want to try it, set `rocket_strafe` to 1). If enabled, you'll find that only lateral and vertical velocity would be added (strafing and jumping/falling). After all, rockets are powered, so how fast the player is moving forward and back is ignored. We know this isn't entirely realistic but if forward/back velocity was added, rockets would be noticeably slower/faster and would look rather silly. This is because the player is really not that much slower than a rocket, at least compared to how fast a real person would be relative to a real rocket. Anyhow, if you enable this, and you're strafing while you fire a rocket, it's going to strafe too. This makes firing rockets while in motion quite a bit, umm, different than what you're used to. **Let us know** what you think.

Rotation of brush models as they affect other entities

Stand on a rotating brush in the regular game, and you'll ride it, but you won't turn with it. Drop an item on that rotating brush model and it will go round and round, but won't turn either. **Lazarus** offers the option to make riding

entities rotate with the rotating bmodel. This is an option available to new maps only - it is **not** a global change. This is because we didn't like the idea of changing the effect so drastically for existing maps. So instead, **Lazarus** offers a new key - **turn_rider**. This key is a property of rotating bmodels and can be enabled or not, as desired. This also allows for the option to be set for individual rotating bmodels... in the same map, one rotating brush might be set to turn its rider while another may be set not to. Turn_rider, when used, affects the player, monsters, items, and anything else that might be riding on the rotating brush.

Monster mass

The effect of mass is not changed in the game; but there is a way to change what you see happening. **Lazarus** allows for the mass of **monsters** and **actors** to be adjusted. Effects such as the knockback from being hit with weapon fire are determined by the monster's/actor's mass. You may have noticed an upward shotgun blast can send a grunt flying while a tank stands firm and takes it; this is because of their differing mass values. While the defaults, monster-by-monster, are pretty appropriate, you can change them if you don't like how they're getting thrown around or not getting thrown around in your map. This is actually a nice tweak for a map that uses a non-standard gravity.

Probably this ability to adjust mass is much more relevant to the **misc_actor**; some player models look pretty beefy, yet the default actor mass is pretty low. It seems silly for a single shotgun blast to throw a big bruiser right off the edge of a cliff, so you might want to increase the mass of the actor to make him more sure-footed. On the other hand, a platoon of very lightweight monster_soldier's can be very amusing too.

[Lazarus Main Page](#)

Pickups

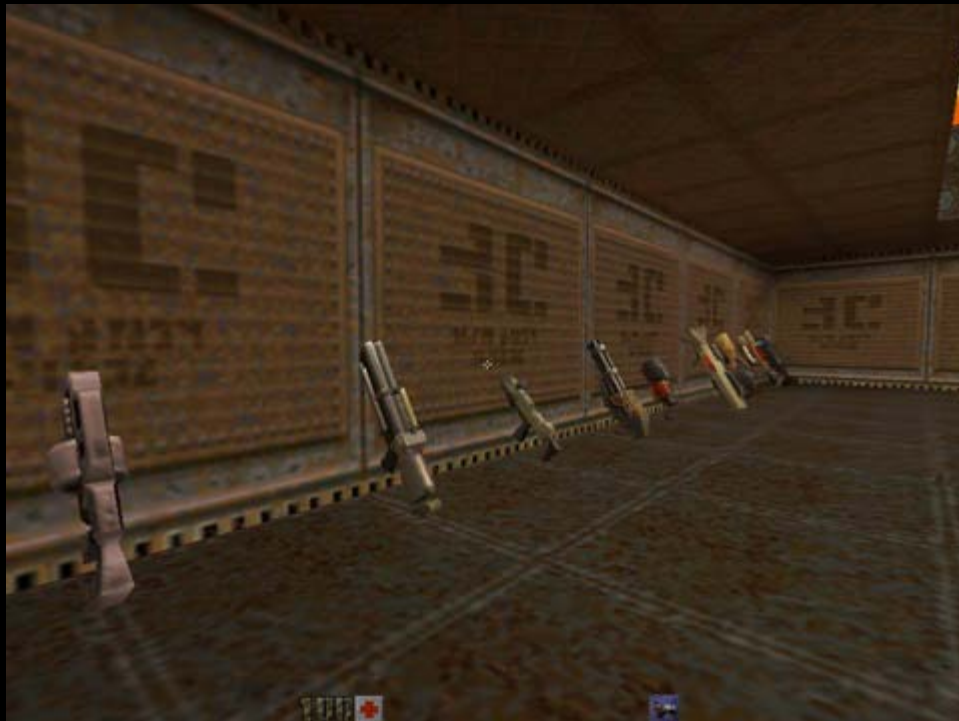
Movewith. This is the targetname of the parent entity you want the pickup item to **movewith**.

TRIGGER_SPAWN spawnflag (=1) (Keys only). Coupled with NO_TOUCH and NO_STUPID_SPINNING (and possibly NO_DROPTOFLOOR), you'll be able to place keycards that appear to be dropped into slots or placed in some other configuration, in much the same way that key_powercube is used.

NO_TOUCH spawnflag (=2) (Keys only). If set, the key cannot be picked up.

NO_STUPID_SPINNING spawnflag (=4) for all pickup items other than health. This is useful if you would like to use the "angles" values to place pickups at an odd orientation. This setting **is** preserved through respawns in deathmatch. There's nothing we can do, however, about dropped items... they will continue to spin as with normal Quake2.

NO_DROPTOFLOOR spawnflag (=8), again for all pickup items other than health. If set, the item will remain wherever it was placed in the map. You give up error diagnostics with droptofloor errors, but as long as you place the pickups such that the center of the bounding box is not embedded in a brush, everything will work out fine. Coupled with NO_STUPID_SPINNING and the angles values, you can place weapons in just about any orientation and any position that you like. For example:



One thing you should be careful of with NO_DROPTOFLOOR: It is now possible to place a pickup item such that its bounding box extends through a wall. The item will then be accessible from either side of the wall, which probably isn't what you intended.

SHOOTABLE spawnflag (=16) and **health** key. If set, you can shoot and destroy any pickup item. The downside to using this is that the item must be made solid. This is no problem if you can pick the item up... there will be a barely noticeable bump as you run into the model. But if you're already stocked up on the item, you won't be able to pick it up and the item will block your path... not that this is a serious problem, but it's not what we're accustomed to. Actually this feature might be put to use in a really nasty scenario - say a player's escape route is blocked by excess health. Does he stay and fight or destroy health he might later need? And... this is no lie... my pal Mad Dog's young son came up with this scenario: Player walks into a room where a target_anger causes monster to attack all the goodies... kids these days are so sweet, don't you think?

In this version, destroyed items are respawned in 30 seconds in deathmatch. This will likely go through some tweaking.

Lazarus Main Page

Point_combat

The **Lazarus** point_combat is for the most part identical in operation to the standard Quake2 point_combat point entity, but includes a couple of coding improvements. The original game caused a point_combat's targetname to be destroyed after its first use. This rendered it impossible to re-use a point_combat, while at the same time it was allowed to continue to exist as an active edict.

Lazarus removes the targetname-destroying effect, allowing for a point_combat to be re-used. Additionally, **count** support is added, which allows it to be auto-killtargeted. When count is 0, the code internally changes that value to 1. The effect of this is that any standard Q2 map run under **Lazarus** would see its point_combats function in the same way: They would only be used once. But instead of the entity hanging around afterwards as an active edict, it would be removed from the map after its first use.

So, if you want to use a point_combat multiple times, assign a **count** value of greater than 1. If for some reason you want the point_combat to be persistent, and never disappear, use a count value of (-1).

Lazarus also provides a new **DRIVE_TRAIN** spawnflag for point_combat. If placed near the driving position of a **func_tracktrain**, when a monster touches this point_combat he becomes the driver of the train (if the train doesn't already have a driver). The monster will never dismount the train once he takes charge of it.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

Specifies the number of times the point_combat will be used before it is auto-killtargeted (see [this page](#) for details).

If count=0, the code will internally change that value to 1.

If count=(-1), no auto-killtargeting will occur.

pathtarget

Targetname of the entity to be triggered when the point_combat is used.

target

Targetname of the next point_combat in the path.

targetname

Name of the specific point_combat.

Spawnflags

HOLD Spawnflag (=1)

The monster or actor which uses the point_combat will stand in place and not run around.

DRIVE_TRAIN

If set, once a monster touches this point_combat the code will check to see whether he is in a good driving position for a **func_tracktrain** (and if the point_combat is placed properly this should always be true. If it is, the monster becomes the driver of the train, and the train accelerates to full speed.

NOT_IN_EASY Spawnflag (=256)

The point_combat will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The point_combat will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The point_combat will be inhibited and not appear when skill=2 or greater.

[Lazarus Main Page](#)

Lazarus Redistribution



So you want to make a Lazarus map...

...and you have a **Lazarus** game folder now. So does that mean that users who download your maps should have a **Lazarus** game folder too? Not necessarily. Should you advise users to place your creations in a **Lazarus** game folder so as to run it? *No - we strongly advise against this.*

There's a fine conceptual point that we'd like to make clear: When you look at the **Lazarus** game folder on your computer, you're looking at a development folder, not a gameplay one. This isn't CTF or AQ. So don't expect to see a **Lazarus** GameSpy tab. Maybe in the future we'll come up with a new folder name for this purpose. But for now... nah.

At this moment in time we're focusing on the single-player side of things, since that's where the entity fun is. It's turned out that some of these new entities are OK to use in multiplayer games, but many aren't so good, for bandwidth reasons. So we're not really expecting to see a slew of **Lazarus**-based multiplayer maps appearing... if you feel hell-bent on making one then **let us know** and maybe we'll think about this some more. For those of you thinking about making a *single-player* map using the **Lazarus** game .DLL, read on:

Single Player Distributions

While the following may seem a bit intimidating (well long-winded anyway), if you follow the steps described, you'll end up with a distribution that will be easy for the casual user (read: don't know nothin' about level editing) to install and run your maps.

1. **Setting up Your Game Folder**
2. **Determining What You Need, and What You Don't**
3. **Testing the Distribution**

Setting up Your Game Folder

While you're constructing your maps, it's totally fine (and expected) that you'd be running them with +set game lazarus (see [this page](#) if you don't know what we're talking about here). But when your maps reach their final stages, it's time to set up your own game folder, which will be the one you'll advise users to utilize when running your maps. Call it whatever you want. The usual thing to do is to name it after your nick, or name it after the title of your map unit. Just make sure it's a subfolder of **Quake2**, just as BASEQ2 is. You'll now be running your maps with the command +set game myfolder, where "myfolder" is the name of, umm, your folder.

Determining What You Need, and What You Don't

Now you'll want to assemble the files needed to run your maps, and place them in your new game folder. What files you'll need requires a bit of thought and care, since what you should want to do is to include every file that will be needed, but not include any that aren't (so as to keep the size of the download as small as possible, out of consideration for those of us who still suffer with glacial connect speeds).

Please note that redistribution of any and all files found here on the [Lazarus website](#) are subject to the [Terms of Distribution](#) outlined below.

Below is a detailed listing of the **Lazarus** files available on the [downloads](#) page, and their importance is designated thusly:

- **Essential Files** - Files you *must* include!!
- **Additional Files** - Files you *may or may not need*, depending on the entities you've elected to use.
- **Optional Files** - Files that are *completely optional*, but we think they're kinda cool so we'd like to see them used.
- **More Optional Files** - Files that are also *optional*, that simply happen to appear in the *Lazarus* example maps.
- **The ActorPak** - The player models, skins and sounds that can potentially be used by misc_actors.

Simple, right? Everything in **white** you have to have, and everything in **blue** is a case-by-case sort of thing. Anything else is by your discretion. Now the files in detail:

Essential Files

GAMEX86.DLL

This file is the guts of **Lazarus**.

SUGGESTED.CFG

Facilitates adding the new key bindings that **Lazarus** uses. This file should *not* be imported into a pakfile, since it's desireable to leave it easily accessible to the user to edit as he wishes.

PAK0.PAK

Contains only the files that can be called when running *any* map under a **Lazarus** game. They are:

pics/zoom.pcx	The crosshair pic for the sniper zoom.
default.cfg	Replaces the game's stock default.cfg so as to avoid a few problems no one need have to deal with. Also allows for the opportunity to tweak Lazarus-specific defaults as desired.

MAP1.BSP, MAP2.BSP, etc

Of course, these files are your maps. This is the point of all this after all.

TEXT FILE

This is a readme-style file where the level maker typically talks about what went into making his maps, gives a background story for them, notes known problems or bugs if any, offers a few word of thanks here and there, and most importantly, gives instructions on how to run the maps. If you need help writing such a file, then check out [.texGen](#). We also have a few requirements as to the contents of this file, which we would appreciate you [reading](#).

In single-player distributions, you should include your maps and any custom sounds, textures, etc (if any) in a pakfile anyhow for the convenience of the user. So, the easiest thing to do is to use a copy of the **Lazarus** PAK0.PAK as a base for building your own pakfile, adding to it as needed. Need a PAK editor? Go [here](#) to get the one we use.

Additional Files

PAK1.PAK

What you'll need out of this pakfile depends entirely on which **Lazarus** entities you've elected to make use of, and in some cases what spawnflag or keyvalue you've set for them. If you need any of these files, extract them from PAK1.PAK and import them into the PAK0.PAK you're building for your own game folder. If you don't need a file, then don't extract it.

models/cable/tris.md2 models/cable/skin.pcx	Crane cable model & skin. Used by: crane entity group (all)
models/items/f_light/tris.md2 models/items/f_light/skin.pcx	Flashlight pickup model & skin. Used by: item_flashlight (all)

models/items/jet/tris.md2 models/items/jet/skin.pcx pics/p_jet.pcx sound/jetpack/activate.wav sound/jetpack/rev.wav sound/jetpack/revrun.wav sound/jetpack/running.wav sound/jetpack/shutdown.wav sound/jetpack/stutter.wav	Jetpack icon, model, skin, & sounds. Used by: item_jetpack (all)
sound/train/X/speed1.wav sound/train/X/speed2.wav sound/train/X/speed3.wav	Train sounds. Used by: func_tracktrain (<i>X</i> = <i>sounds</i> value; none required for <i>sounds</i> < 0)
models/items/ammo/fuel/medium/tris.md2 models/items/ammo/fuel/medium/skin.pcx pics/a_fuel.pcx	Fuel ammo icon, model & skin. Used by: ammo_fuel (all)
models/items/ammo/homing/medium/tris.md2 models/items/ammo/homing/medium/skin.pcx models/weapons/v_homing/tris.md2 models/weapons/v_homing/skin.pcx pics/a_homing.pcx	Homing missile icon, model & skin and viewmodel and skin for rocket launcher. Used by: ammo_homing_missiles (all)
models/objects/drop/tris.md2 models/objects/drop/skin.pcx	Raindrop model used with target_precipitation (style=0)
models/objects/drop/heavy.md2 models/objects/drop/skin.pcx	Raindrop model used with target_precipitation (style=1)
models/objects/snow/tris.md2 models/objects/snow/skin.pcx	Snowflake model used with target_precipitation (style=2)
models/objects/leaf1/tris.md2 models/objects/leaf1/skin.pcx models/objects/leaf2/tris.md2 models/objects/leaf2/skin.pcx models/objects/leaf3/tris.md2 models/objects/leaf3/skin.pcx	Leaf models used with target_precipitation (style=3)
models/objects/rock1/tris.md2 models/objects/rock1/skin.pcx models/objects/rock2/tris.md2 models/objects/rock2/skin.pcx	Rock models & skins. Used by: target_rocks (all)
pics/speed0.pcx pics/speed1.pcx pics/speed2.pcx pics/speed3.pcx pics/speedr1.pcx pics/speedr2.pcx pics/speedr3.pcx	Forward/reverse indicators. Used by: func_tracktrain (except NO_HUD spawnflag) func_vehicle (all)
sound/engine/engine.wav sound/engine/idle.wav	Engine sounds and forward/reverse indicators. Used by: func_vehicle (all)
pics/textdisplay.pcx	Text message background plaque. Used by: target_failure (except NO_BACKGROUND spawnflag) target_text (except NO_BACKGROUND spawnflag)

sound/mud/mud_in2.wav sound/mud/mud_out1.wav sound/mud/mud_un1.wav sound/mud/wade_mud1.wav sound/mud/wade_mud2.wav	Player mud movement sounds. Used by: func_bobbingwater (MUD spawnflag) func_water (MUD spawnflag)
sound/weapons/homing/lockon.wav	Homing rocket target lock-on sound. Used by: monster_boss2 (HOMING spawnflag) monster_chick (HOMING spawnflag) monster_supertank (HOMING spawnflag) monster_tank (HOMING spawnflag) monster_tank_commander (HOMING spawnflag) target_blaster (sounds=4) turret_breach (sounds=4)
sprites/point.pcx sprites/point.sp2	Invisible sprite used in place of a model. Used by: crane_hook (SPOTLIGHT spawnflag) model_spawn (NO_MODEL spawnflag) model_train (NO_MODEL spawnflag)
models/items/stasis/tris.md2 models/items/stasis/skin.pcx sound/items/stasis.wav sound/items/stasis_start.wav sound/items/stasis_stop.wav	Model and sounds required by item_freeze .
models/objects/barrel_gibs/*. models/objects/crate_gibs/*. models/objects/crystal_gibs/*. models/objects/glass_gibs/*. models/objects/mech_gibs/*. models/objects/metal_gibs/*. models/objects/rock_gibs/*. models/objects/tech_gibs/*. models/objects/wood_gibs/*.	Models and skins used by func_explosive and func_pushable with non-zero gib_type values, and by damageable (health < 0) func_door and func_door_rotating with non-zero gib_type values.

FMOD.DLL

This .dll is only necessary if you're using a **target_playback** or custom footstep sounds, since without it, the custom sound files you wish to play will not play. Of course you'll have to include the custom sound files too.

Optional Files

PAK2.PAK

If you don't want to use any of these, you certainly don't have to. But we've grouped them in this PAKfile since we either like them or we think they could be pretty useful, depending on what you've got going in your maps. Again, if you want to use any of these files, extract them from PAK2.PAK and import them into the PAK0.PAK you're building for your own game folder. If you don't want a file, then don't extract it.

sound/weapons/grenlf1a.wav	Grenade launcher firing sound. Also used when a player joins a multiplayer game.
sound/weapons/rocklx1a.wav	Rocket explosion sound. Also used for destroyed entities and all airborne explosions.
sprites/lightning.sp2 sprites/lightning0.pcx sprites/lightning1.pcx sprites/lightning2.pcx	Lightning sprite that you might find useful with a model_spawn . This sprite has 9 animation frames. For usage see the <i>rain</i> example map.

sprites/lightning3.pcx sprites/lightning4.pcx	
textures/crane/panel.wal	Useful texture for the crane_control .

More Optional Files

PAK3.PAK

The files grouped in this PAKfile are the textures and environment maps that appear in the **Lazarus** example maps. If you'd like to use them, that's fine with us.

DEVELOPER.CFG & NORMAL.CFG

This configuration file pair allows for the level maker to go easily from mapedit mode to gameplay mode and back again. We don't recommend that you include these files, mainly because you might have to explain what they do :-)

The ActorPak

PAK4.PAK

The files grouped in this PAKfile are the modified player models, skins and sounds which can potentially be used by **misc_actors**. If your map(s) includes actors which reference any of these, then you'll definitely need them - but you should see the appropriate author's permission before distributing your work.

Player Sounds

PAK5.PAK

This pak file contains duplicates of the player sounds for the models in PAK4.PAK. This file is **only** necessary if you want end users to be able to use the PAK4.PAK models as their own player model. Why the duplication of effort? Player model sounds in PAK4.PAK are located in `sound/../../players/<modelname>` rather than simply `players/<modelname>`. While this is necessary to use those sounds with `misc_actor`, this is of course not the correct location for "real" player models.

Testing the Distribution

At this point you should have a fully assembled PAKfile called PAK0.PAK (which includes your maps and all files that they will call), the **Lazarus** GAMEX86.DLL, SUGGESTED.CFG, and a readme-style TEXT FILE all sitting in your own little game folder. Its integrity needs to be tested now.

Restoring Your Quake2 Installation

What works on your system may not work on the system of the user who downloaded your map(s). Most level makers accumulate a host of textures, environment maps, sounds, etc. For convenience, these assorted files tend to be placed in subfolders of \BASEQ2\. This practice tends to mask potential problems of necessary files being omitted from the distribution. Your maps may run fine, not because your distribution includes the needed files, but because when run on your system, QUAKE2.EXE found the needed files lying around loose in \BASEQ2\. So the first thing you have to do is to clear out all such files, including any user-created PAKfiles, from the BASEQ2 folder, and bring yourself back to what things looked like in there when you first installed (and fully patched) Quake2.

Testing For Missing Files

Once that's done, fire up the game using `+set game myfolder`. Don't load any maps yet; you need to set a few console variables first. Bring down the console and type:

```
readout 1
allow_download 1
allow_download_sounds 1
```

```
allow_download_models 1
allow_download_players 1
allow_download_maps 1
```

Lazarus suppresses the irritating "refusing to download a path with .." messages for single-player games by forcing **allow_download** to 0. This also defeats the purpose of what we're trying to do here. You **must** set **readout** to **1** so that the allow_download setting will stick.

Now, you really don't need to set all of these allow_download_* variables, but better safe than sorry. Now load **each** of your maps **individually** from the console, and pay close attention to any messages that say something like "server does not have this file". This means there's a file the map called, and the file wasn't found. Take note of what file it was that was missing, and get it in your PAKfile to correct this.

If everything loads up nice and clean, with no errors, then you have everything you need, and you should be ready to zip it all up. But not so fast - there's one more thing you can attend to, and it's something that is almost universally overlooked: Customizing **default.cfg**.

Customizing Default.cfg

If you wish, you can further polish things by extracting **default.cfg** from the PAKfile and changing the NewGame alias so it will load the first map in your unit, rather than loading base1.bsp. For example, you could change the string to read:

```
alias newgame "killserver;maxclients 1;deathmatch 0;map mymap"
```

Just as the regular game does, you may choose to run a cinematic first (map *ntro.cin+mymap) or a demo (map *mydemo.dm2+mymap). Of course, the map that the demo was recorded on should be included as well or the demo won't run. If you want to use a demo, be *sure to record the demo while running the game in your game folder*. This is a requirement of using a demo as an introductory sequence, since loading demos also sets the value of "game". And that value had better be the name of your game folder, or the **Lazarus** code may not be used when running the map that loads after the demo.

Customizing default.cfg serves another purpose. We've added some changes to monster behavior, third-person view when moving pushable objects, etc... and some of these things may not be to your liking. Or they may be inappropriate for your maps. We've provided cvars (console variables) which can serve to turn some of these things off for this very reason. These cvars are set in default.cfg, and you can change their settings there. The **Lazarus** default.cfg is commented so you can easily know what to change and what to leave alone. Be sure and check it out.

You can also use default.cfg to set non-**Lazarus** console variables to something other than their defaults, but there are problems with this - if you're trying to set something that's already defined in the user's \BASEQ2\ config.cfg file, well then your intended change isn't going to stick. Also, if it's something that may be defined in the user's \BASEQ2\ autoexec.cfg file, then the intended change again won't stick. Do **not** try and get around this by including a custom autoexec.cfg - this just pisses people off.

There is a way to kinda sorta set what you want anyway... you could exec a custom config file in the above-mentioned **newgame** string. Just don't try and set hardware-specific stuff, or you can expect some pretty nasty e-mail. This really isn't foolproof either, since it won't work if the player quits your maps after playing awhile and then continues later by loading a saved game. If you're unfamiliar with configs and scripts, then you could always check out some stuff [here](#) (*a little shameless plug here from MD, sorry*).

Whatever you decide to do with the "newgame" alias, make sure to test it by loading a new game using the in-game menu. When you have this working fine, import the new **default.cfg** back into your PAKfile and test it again - PAKfiles can corrupt, you know...

Wrapping (and zipping) It All Up

Once all checks out, the contents of your game folder need to be compressed into a single file to facilitate file transfers. A .ZIP file is fine; you can get **WinZip** for this purpose. You may wish to utilize an installer to avoid the problem of users not being able to open .ZIP files; just be aware there are the paranoid among us who are leery of clicking on unknown .EXE's. One good compromise is to let WinZip

turn the .ZIP into a self-extracting file... since self-extracting files can be optionally opened with WinZip rather than run - which should satisfy the paranoids. Whatever you decide, you need to test this too! Extract the files and try to run them. If they pass muster, you're done and it's time to post the archive somewhere where people can download it, and announce to the world that you've made some really cool maps.

Terms of Distribution

All files, including but not limited to the gamex86.dll, all maps, textures, prefabs, documentation, etc. available here at the [Lazarus website](#) are the intellectual property of the principals of the **Lazarus Quake2 Mod** team. We're doing this for fun, and out of a sense of contributing to the ongoing phenomena that is the Quake community. Sadly, we also realize that there are some llamas out there that like to pawn off other's work as their own. Since we take a pretty dim view of this sort of thing, we require that any distribution meet the following conditions:

- **Lazarus** files are to be redistributed only as part of a level maker's map distribution. The files may not be redistributed by themselves.
- A text file must accompany the level maker's distribution. Two copies of this text file should be present: One copy as a "loose", easily accessible file, and the other copy placed within the root directory of the level maker's PAKfile. If multiple PAKfiles are used, there should be a copy of the text file in each one. The text file must include:
 1. Prominent identification of the use of the **Lazarus** gamex86.dll in the "credits" section.
 2. The use of any other **Lazarus** files should be mentioned as well. It isn't necessary to create a detailed list; for example, merely mentioning that **Lazarus** models, skins, textures, etc. were used would suffice.
 3. The current URL to the **Lazarus** website, as of the time of the level maker's distribution, should also appear. Our current home is <http://planetquake.com/lazarus/>. If the website is closed and there is no new site, this provision is not applicable.
- None of the documentation may be redistributed for any reason.

Please accept our apologies for feeling the need to bum you out with the above. Now go make some maps!

DISCLAIMER

Neither David Hyde nor Tony Ferrara make any warranties regarding this product. Users assume responsibility for the results achieved for computer program use and should verify applicability and accuracy.

[Lazarus Main Page](#)

Sniper zoom

A fully adjustable sniper mode allows you to smoothly zoom in/out between the default field-of-view and 5 degrees (or anywhere in between). It is also possible to snap instantly between normal and the last used sniper setting as well. The `suggested.cfg` file, which is included with the **Lazarus** download, will bind the controls you'll need to 3 keys: **Z**=zoomin, **X**=zoom, and **C**=zoomout.

- **+zoomin** - when pressed, sniper mode is active and fov will decrease smoothly until the key is released, or the fov reaches 5 degrees.
- **+zoomout** - when pressed, sniper mode is active and fov will increase smoothly until the key is released, or the fov is equal to the user's default fov.
- **+zoom** - toggles the view between the default fov and the last fov set when **+zoomin** or **+zoomout** keys were released. Yes that means it "remembers" your last zoom magnification.

There's also a couple of bits going on behind the scenes that you might be interested in:

- **zoomrate** - persistent cvar controlling the speed of **+zoomin** and **+zoomout**. Default value is 80 degrees/second, but you can change this at the console if you wish a faster or slower rate.
- **zoomsnap** - cvar specifying the fov to snap to with **+zoom**. Initially set to 20 degrees, but this value is updated every time **+zoomin** or **+zoomout** are released (unless a **+zoomout** operation takes you to the default fov).

These, and more **sniper zoom** commands can be found on the [console commands](#) page.

As outlined in the [redistribution](#) doc, the sniper zoom requires the image file for the crosshair:

- [pics/zoom.pcx](#)

The crosshair used by Lazarus is a slightly modified version of one created by Madman. You can check more of his and others' crosshair creations at www.bullseyecrosshairs.com.

Note: **Lazarus** adjusts mouse and joystick sensitivity while zooming to give the player roughly the same control he has with the normal fov. This feature comes at a price, however: There's really no good way to determine or change a client's mouse and joystick sensitivities without adding a bandwidth burden, so... we've decided to disable zoom in multiplayer games, at least for now.

New Sounds

Lazarus currently includes the following new sounds, which replace those used in the standard game:

weapons/grenlf1a.wav

Grenade launcher firing sound. This also replaces the sound of a player entering a multiplayer game.

weapons/rocklx1a.wav

Rocket explosion sound.

The following sounds are also new, but will only be used when using one of the new/modified **Lazarus** entities which calls them:

sound/engine/engine.wav

sound/engine/idle.wav

Engine sounds used by the **func_vehicle**.

sound/mud/mud_in2.wav

sound/mud/mud_out1.wav

sound/mud/mud_un1.wav

sound/mud/mud_un2.wav

sound/mud/wade_mud1.wav

sound/mud/wade_mud2.wav

sound/mud/wade_mud3.wav

Player movement sound while in mud, used by **func_water** and **func_bobbingwater**, when they are used as, err, mud. *Sound files courtesy of Maulok (Anthony Bouvette).*

sound/weapons/homing/lockon.wav

Homing rocket lockon sound, used when the player is targeted by any **monster**, **actor**, **turret**, or **blaster entity** which is set to use homing rockets.

We are always on the lookout for any good royalty-free sounds to replace or add to the standard game sounds, so expect this list to grow. If you would like to make a contribution to this effort, then feel free to **contact us**.

Target_actor

The **Lazarus** target_actor is very similar to the standard Quake2 target_actor point entity. However this entity was originally designed to function as a path node which imparted its scripting instructions to the **misc_actor** entity. Since the misc_actor code was incomplete, the usefulness of the target_actor was pretty limited. **Lazarus** changes that by bringing the misc_actor to life.

The target_actor is kind of a souped-up **path_corner**, since it is also a pathing node. If you wish to use the scripting capability of the target_actor without the pathing element, check out the **target_anger**.

Lazarus also adds **count** support to the target_actor, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

When non-zero, specifies the number of times the target_actor will be used as a path node before being auto-killtargeted (see [this page](#) for details). Default=0.

height

Specifies the vertical velocity in units/second given to the misc_actor/monster_* which uses the target_actor. Default=200.

message

Message to be printed to the screen when the target_actor is used.

pathtarget

Targetname of the entity the misc_actor/monster_* will act upon when the target_actor is used.

speed

Specifies the horizontal velocity in units/second given to the misc_actor/monster_* which uses the target_actor. Default=200.

target

Targetname of the next target_actor/path_corner in the path.

targetname

Name of the specific target_actor.

wait

Specifies the amount of time in seconds that the misc_actor/monster_* which uses the target_actor will pause before proceeding to the next target_actor/path_corner in the sequence. If wait=(-1), the misc_actor/monster_* will not proceed unless it is triggered to move again. Default=0.

Spawnflags

JUMP Spawnflag (=1)

The target_actor will cause the misc_actor/monster_* to jump.

SHOOT Spawnflag (=2)

The target_actor will cause the misc_actor/monster_* to shoot once at its pathtarget.

ATTACK Spawnflag (=4)

The target_actor will cause the misc_actor/monster_* to shoot and attempt to kill its pathtarget.

HOLD Spawnflag (=16)

The target_actor will cause the misc_actor/monster_* to stand in place and not run around.

BRUTAL Spawnflag (=32)

The target_actor will cause the misc_actor/monster_* to shoot, attempt to kill, and gib its pathtarget.

NOT_IN_EASY Spawnflag (=256)

The target_actor will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_actor will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_actor will be inhibited and not appear when skill=2 or greater.

Target_anger

The **Lazarus** target_anger point entity first appeared as an entity straight from the Rogue mission pack. That entity could be used to make monsters attack virtually whatever you wanted them to. Its current form now incorporates many elements of the **target_actor** scripting entity, to add flexibility in using the **misc_actor**. The result is a very versatile scripting entity that can be triggered by almost anything to make your monsters and actors do almost anything.

The limitation of using a path node like a target_actor to handle scripting chores is that when the monster/actor is put into motion, he'll follow his programmed path the same way each time. With a target_anger, you have the choice of whether or not to trigger it, or even a choice of triggering one of many. For example, a room with 3 entrances which is guarded by a single monster could have a trigger at each entrance, and each trigger could target a different target_anger. Each target_anger could direct the monster to do something different. As another example, a RANDOM **target_rotation** that is triggered only one time can target any one of a number of different target_angers (all of which point to the same monster), so that every time a map is played the monster could have a different reaction.

Monsters and actors can be made to attack a brush model as well (blow up an explosive object, shoot out a window, etc) - but they need to know where it is, or they'll assume the origin of the bmodel is the map origin (0 0 0). Give your bmodel an origin brush and if directed to do so, they'll attack it as you intend them to. Their target will be the center of that origin brush, so you can place it exactly where you want them to shoot.

Version 1.8 and later versions add one important change to target_anger that might be useful for scripted monster battles (or murdering hapless misc_insanes). Rather than automatically selecting the first found *killtarget*, target_anger will select the closest matching entity with health > 0 for each *target* assassin. So, for example, to make a monster murder a large group of misc_insanes, give all misc_insanes the same targetname, start the action with a target_anger, and set the deathtarget of each misc_insane to the targetname of the target_anger. Now the death of one misc_insane will result in the monster becoming angry at another misc_insane, and the sequence will be repeated until all misc_insanes are dead or the monster is distracted by some other event (such as being shot by the player).

Key/value pairs

angle

Sets the direction on the XY plane that the targeted misc_actor/monster_* will move. This key is only relevant if **speed** is non-zero and the actor/monster is not already moving.

count

When non-zero, specifies the number of times the target_anger will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

height

Specifies the vertical velocity in units/second given to the targeted misc_actor/monster_*. Default=0.

killtarget

Targetname of the entity the targeted misc_actor/monster_* will act upon.

pathtarget

Targetname of the entity the targeted misc_actor/monster_* will move to. Incompatible with **HOLD**.

speed

Specifies the horizontal velocity in units/second given to the targeted misc_actor/monster_*. Default=0. Direction of movement is the same as the current movement direction of the monster/actor. If the monster/actor is not moving, then direction is determined by **angle**.

target

Targetname of the misc_actor/monster_* the target_anger is to act upon.

targetname

Name of the specific target_anger.

Spawnflags

HOLD Spawnflag (=16)

The target_anger will cause the misc_actor/monster_* to stand in place and not run around.
Incompatible with **pathtarget**.

BRUTAL Spawnflag (=32)

The target_anger will cause the misc_actor/monster_* to shoot, attempt to kill, and gib its **killtarget**.

NOT_IN_EASY Spawnflag (=256)

The target_anger will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_anger will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_anger will be inhibited and not appear when skill=2 or greater.

Lazarus Main Page

Target_animation

The **Lazarus** target_animation point entity is a mechanism which is used to trigger an animation sequence in another entity, such as a **misc_actor**. You can easily select a number of the canned sequences used by player models (such as taunts) or you can select your own series of frames to play. If a misc_actor is in the middle of a battle, the animation sequences called by his actions will override those specified by a target_animation which targets him.

The **ACTIVATOR** spawnflag is designed to filter the action of the target_animation, by causing it to check what entity activated it. (An entity which sets a triggered event in motion is considered the "activator" of that event). When such a target_animation is triggered, it checks the entity class of the activator against the entity class specified by the **message** key. If the activating entity's class equals the value of "message", then the target_activator will fire. If the activating entity's class does not equal the value of "message", well then it won't fire.

ACTIVATOR can be used in another way - to cause the activating entity to animate itself. Suppose you wanted **any** misc_actor which happens to kill a monster to play a set of animation frames. To accomplish this, define **message** as "misc_actor", and leave the **target** key **undefined**. The code causes a target_animation with no target set to point to the activator instead (assuming the **message** filter is satisfied, of course).

Just in case you're looking to trigger a target_animation by using a path_corner pathtarget, here's a bit of info that may keep you from frustrating yourself: Make sure that such path_corners do not use "wait" values greater than 0. If the path_corner's wait>0, the target_animation will simply not be triggered when the path_corner is touched.

Key/value pairs

count

When non-zero, specifies the number of times the target_animation will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

framenumbers

Specifies the number of frames to play after **startframe**. Default=0. Ignored unless **sounds**=0.

message

Specifies the entity class that must activate the particular target_animation before it will fire. If **target** is undefined, the target_animation will animate the activator itself. Ignored if **ACTIVATOR** is not set.

sounds

Specifies a canned player model animation sequence. If sounds=0, then **startframe** and **framenumbers** values are used. Note that these sequences are applicable **only** to misc_actor. Although the code won't prevent you from using these values for other entities, at the very least you'll end up with very goofy animation sequences, and very likely produce a stream of error messages if you use a non-zero value for any entity other than misc_actor. Default=0. Possible choices:

- 0: Use startframe/framenumbers
- 1: Jump
- 2: Flipoff
- 3: Salute
- 4: Taunt
- 5: Wave
- 6: Point

startframe

Specifies the first frame number to be played. See also **framenumbers**. Default=0. Ignored unless **sounds**=0.

target

Targetname of the entity to be animated. If "target" is not defined, the target_animation will target its

ACTIVATOR.

targetname

Name of the specific target_animation.

Spawnflags

ACTIVATOR Spawnflag (=1)

The target_animation will only fire if the "activating" entity is the same as the one defined by **message**.

NOT_IN_EASY Spawnflag (=256)

The target_animation will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_animation will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_animation will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_animation will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Target_attractor

Target_attractor is an entity magnet gizmo. It pulls the target entity towards it, and may optionally trigger another entity once the target reaches the attractor. Target_attractor works by continuously adjusting the target entity's velocity vector to point towards the attractor. The target entity is given a minimum *speed* value to help move it towards the attractor. This entity is useful for more than the obvious player trap; you might also use it as a lifting device or to assist in the exact placement of gravity-affected brush models like the [func_pushable](#) and the [func_object](#).

Key/Value Pairs

Targetname

Name of the target_attractor. If the [START_ON](#) spawnflag is not set, **or** if you want to toggle the target_attractor on/off, it must have a targetname.

Target

Specifies the targetname of the entity to attract. This value is ignored if the [PLAYER](#) or [MONSTER](#) spawnflags are set.

Distance

Maximum range to the target. Entities beyond this range will not be attracted. If not used or 0, distance is ignored and all targets will be attracted.

Speed

Minimum speed to pull the target with. Note that if the target is normally affected by gravity and the target_attractor pulls the target **up**, *speed* must be greater than the *sv_gravity* value * 0.1, 0.1 being the time interval between target_attractor "pulls". For normal gravity (800 units/sec/sec), this means speed must be >80 to pull up a player or monster or other entity that is affected by gravity. Alternatively, you can set the [NO_GRAVITY](#) spawnflag to temporarily turn gravity off for the target entity.

Use a negative value for speed to repel rather than attract targets. For negative *speed* values, [accel](#) should also be negative or 0.

Players attracted by a target_attractor with the [PLAYER](#) spawnflag **can** run away from it, depending on the *speed* value and whether the attractor is pulling the player up off the floor (in this case, the player is immediately pulled from the floor, and running will have no effect unless gravity helps him out a bit).

Accel

Acceleration value. Attraction speed will increase by *accel* units/sec every second. More accurately, it will increase by *accel*/10 units/sec every 0.1 seconds. If not used or 0, attraction speed is constant.

Pathtarget

Entity to trigger once the target reaches the target_attractor. The pathtarget will only be fired once per on/off cycle of the target_attractor, regardless of how many targets are being attracted simultaneously.

count

When non-zero, specifies the number of times the attractor will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

Noise

Sound to play when the target_attractor is active.

Sounds

Effect to use when attracting a target. If sounds is non-zero, the [SIGHT](#) and [SINGLE_TARGET](#) spawnflags are automatically set, whether they were set in the map file or

not.
0 = none
1 = medic cable
2 = BFG laser

Movewith

Name of the func_train or model_train to **movewith**.

Spawnflags

START_ON (=1)

If set, the target_attractor will be active when the map loads. If not set, target_attractor must be triggered to function.

PLAYER (=2)

If set, target_attractor attracts players. The **target** key is ignored in this case. If the **SIGHT** spawnflag is set, the attractor ignores players that are not visible. If the **SINGLE_TARGET** spawnflag is set, the attractor chooses the closest player.

NO_GRAVITY (=4)

Set this spawnflag to temporarily turn gravity off for the targeted entity. **NOTE:** For best results when **PLAYER** is set, **NO_GRAVITY** should be set as well in cases where an attractor is to lift a player off the floor, or in scenarios where the player is in no way intended to have a chance to successfully fight the pull of the attractor. This is because without setting **NO_GRAVITY**, gravity is allowed to continuously fight against the target_attractor. This results in noticeably jerky motion near the attractor's origin.

MONSTER (=8)

Set this spawnflag to attract ALL monsters. The **target** key is ignored if **MONSTER** is set.

SIGHT (=16)

If set, the target must be visible to the target_attractor to be attracted. **SIGHT** is automatically selected if **sounds** is non-zero.

SINGLE_TARGET (=32)

If set, the target_attractor will select the closest visible (if **SIGHT** is set) valid target and only attract it, rather than all valid targets. **SINGLE_TARGET** is automatically selected if **sounds** is non-zero.

Some Additional Information

Various scenarios can be constructed where once the targeted entity arrives at the attractor, it would be desired to have the attractor immediately release it. This can be accomplished by using a trigger_relay. The target_attractor would be set to **pathtarget** the relay, and the relay in turn would target the attractor, turning it off. In practice this will **not** work unless the relay is given a **delay** value of at least **0.1** seconds.

Target_blaster

The **Lazarus** target_blaster is a significant modification over the standard Quake2 target_blaster point entity. The **sounds** key offers multiple ammo choices, and the blaster can now follow **targeted** entities (much like a target_laser). The blaster can also make use of a **SEEK_PLAYER** spawnflag which lets you target the player. There are a number of other tweaks as well, designed to facilitate better use of these mods; the keyvalue and spawnflag descriptions below go into more detail about this.

As with many other **Lazarus** entities, the target_blaster can also be made to **movewith** a parent entity, and includes **count** support, which allows the blaster to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies an aiming direction on the XY plane. Default=0. Ignored if **target** is used, and/or **SEEK_PLAYER** is set.

angles

Specifies an aiming direction in 3 dimensions, defined by pitch and yaw (roll is ignored). Default=0 0 0. Ignored if **target** is used, and/or **SEEK_PLAYER** is set.

count

When non-zero, specifies the number of times the blaster will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

dmg

Specifies the number of damage hit points the blaster will do with each hit. Default=15.

movewith

Targetname of the parent entity the blaster is to **movewith**.

sounds

Specifies what the blaster will fire. Default=0. Choices are:

- 0: Blaster
- 1: Railgun
- 2: Rockets
- 3: BFG10K
- 4: **Homing rockets**
- 5: Bullet
- 6: Grenade

speed

Specifies speed in units/second of the blaster bolt or the initial speed of a grenade (sounds=6). Default=1000.

target

Targetname of the entity the blaster will fire at. If **IF_VISIBLE** is set, the blaster will not fire at its target unless it has a clear shot. If **SEEK_PLAYER** is set, the player is a higher priority target, and therefore the entity specified by "target" will be ignored. If both **IF_VISIBLE** and **SEEK_PLAYER** are set, the blaster will fall back to firing at its "target" when there is no clear shot at the player.

targetname

Name of the specific target_blaster.

wait

Specifies the amount of time in seconds between blaster firings. Default=0. When wait>0, the target_blaster operates more like a target_laser than the normal Q2 target_blaster. Once triggered, it stays on and fires (or attempts to) once every "wait" seconds until toggled off. If **IF_VISIBLE** is set, and the target_blaster fails to fire because it has no clear shot at its target, it then will search for a valid target every 0.1 seconds until it **does** fire. At that point it will wait the specified amount of time before attempting to fire again. (This is designed to foil attempts by the player to avoid the blaster by timing its shots).

Spawnflags**NO_TRAIL Spawnflag (=1)**

The target_blaster will produce the blaster bolt but not the blaster trail. Ignored unless **sounds=0**.

NO_EFFECTS Spawnflag (=2)

Same as **NO_TRAIL** except that dynamic lighting effects will also not be produced. Ignored unless **sounds=0**.

START_ON Spawnflag (=4)

Specifies that the target_blaster will be active at map startup. Only relevant if **wait>0**.

IF_VISIBLE Spawnflag (=8)

Specifies that the target_blaster must have a clear shot at its target before it will fire. Not relevant if **target** is not used and/or **SEEK_PLAYER** is not set.

SEEK_PLAYER Spawnflag (=128)

Specifies that the target_blaster will target the first player it finds, and will consider players priority targets over entities specified by **target**. Whether the blaster will fire or not, or whether it will fire at the player or at its fallback **target**, is subject to whether or not **IF_VISIBLE** is set.

NOT_IN_EASY Spawnflag (=256)

The target_blaster will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_blaster will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_blaster will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_blaster will be inhibited and not appear when deathmatch=1.

Target_bmodel_spawner

The **Lazarus** target_bmodel_spawner is a point entity which allows multiple brush entities to reference a single, common brush model. Technically, it really doesn't spawn brush models; more precisely it spawns brush entities.

If you've made a large map that uses a lot of brush models, you may have already blundered into the Error: Index Overflow map load crash because your map has too many models in it. The maximum number of unique models that can be in any map is 256; however a few slots in the model array are reserved for other functions so it's wise to keep that number down to no more than 250.

If you have 100 monster_infantry guys in a map, while they are 100 unique entities they reference only a single model. If you have 100 func_doors in your map, these are 100 unique (brush) models. Worse, brush models occupy a model slot even when they are inhibited. The target_bmodel_spawner allows you to make 100 func_doors that only use a single model. Of course, all these doors will be identical to the referenced version (as specified by **source**). This means they will all use the same textures as the parent (you *can't* change that), and use the same keyvalues of the parent (you *can* change this), and all these doors will be lit identically (you *can't* change this).

You can also spawn brush entities endlessly using a single target_bmodel_spawner. For example, you could spawn func_pushables at one end of a room, where they are then pushed to the other end of the room to be destroyed. Just be sure not to spawn one where another already exists, or the 2 will be stuck together.

The spawned clone need not remain identical to the referenced parent. The parent is assumed to be oriented at angle=0; giving a value to **angle/angles** will cause the clone to be rotated relative to the parent. So your 100 doors can face different ways. The target_bmodel_spawner's **target** key doesn't make the spawner target anything, and its **team** key doesn't team it with anything; instead they set the new target and team values, respectively, for the clone. Likewise, the **newtargetname** key allows the clone to be given a unique targetname. If you want further changes, like giving cloned func_explosives different dmg values or cloned func_doors different speeds, the unique targetnames they can have facilitates the use of a **target_change** to accomplish this.

The location of the origin of spawned bmodels is dependent on the location of the origin of the parent bmodel. If the origin of the parent is at the map origin (0 0 0), then the origin of the spawned child will be identical to the origin of the target_bmodel_spawner that spawned it. If the parent is offset from the map origin, then the child will be spawned at an identical offset from the bmodel_spawner's origin. If you wish to locate the parent bmodel somewhere other than at the map origin, and don't want to fuss with the offset math, then you can override the use of the map origin reference, and specify your own origin, by giving the parent bmodel an origin brush.

Key/value pairs

angle

Specifies the facing angle of the spawned brush entity on the XY plane, relative to its referenced parent model. Default=0.

angles

Specifies the facing angle of the spawned brush entity in 3 dimensions, relative to its referenced parent model, as defined by pitch, yaw, and roll. Default=0 0 0.

count

When non-zero, specifies the number of times the target_bmodel_spawner will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

newtargetname

Targetname value which will be inherited by the spawned clone. If no value is given to this key, then the clone will not have a targetname.

newteam

Team value which will be inherited by the spawned clone. If no value is given to this key, then the clone

will not be on a team. (This should be used with **START_ON**, so that there's no chance one teammate can be in the process of moving while another teammate is being spawned).

source

Targetname of the brush model entity which is to be used as a model reference.

target

Target value which will be inherited by the spawned clone. If no value is given to this key, then the clone will not have a target.

targetname

Name of the specific target_bmodel_spawner.

team

Team value which will be inherited by the spawned clone. If no value is given to this key, then the clone will not be on a team. (This should be used with **START_ON**, so that there's no chance one teammate can be in the process of moving while another teammate is being spawned).

Spawnflags**START_ON Spawnflag (=1)**

The target_bmodel_spawner will spawn the brush entity clone at map load.

NOT_IN_EASY Spawnflag (=256)

The target_bmodel_spawner will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_bmodel_spawner will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_bmodel_spawner will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_bmodel_spawner will be inhibited and not appear when deathmatch=1.

Target_cd

The target_cd is a new **Lazarus** point entity which, when triggered, causes a predetermined CD track to begin playing. It will not force CD playback if the user has CD music disabled (cd_nocd=1).

Key/value pairs

count

When non-zero, specifies the number of times the target_cd will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

dmg

Specifies the number of times to loop the CD track. Default=1.

sounds

Specifies CD track number to play. Default=2.

targetname

Name of the specific target_cd.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_cd will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_cd will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_cd will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_cd will be inhibited and not appear when deathmatch=1.

Target_change

The **Lazarus** target_change point entity allows for altering the entity properties of another entity. Have you ever wished that you could have an entity functioning one way, and then later trigger a change in the way it functions, or what it triggers, or what it targets, or what it spawns? This entity may provide a solution.

There are only two **required** keys for the target_change, and they are **targetname** and **target**. Naturally, "targetname" is required so you can trigger the target_change, and "target" is required so you can point it to the entity whose properties you wish to alter. Of necessity, the syntax for the values of "target" is a little odd: It will accept a pair of comma-separated values. The first value in the pair for "target" refers to the targetname of the targeted entity, and the second refers to the new target you may wish to assign to that entity.

One other special key exists: The **newtargetname** key is used to assign a new targetname to the targeted entity.

Virtually all other keys are valid, assuming the targeted entity can use them. They are **not listed** below since that would be pretty redundant and pointless - all they do is give the target_change the instruction to attempt to assign new values to the keys of the targeted entity. Likewise, new spawnflags can be assigned as well, but you'll want to make sure the spawnflags total is the same as what you want to see in the targeted entity.

Naturally, there are quite a number of different keyvalues, an even larger number of different entities, and more importantly there's a big difference between changing a keyvalue of an entity that is active and currently referencing that keyvalue versus an entity that is at rest and not currently referencing that keyvalue. So there's no guarantee that all possible changes will work with all possible entities under all possible circumstances. However, plenty of combinations certainly do work; if there's something you'd like to do that could potentially be possible using a target_change, then give it a try.

Key/value pairs

newtargetname

The new targetname value you may wish to assign to the targeted entity.

target

Two values are valid here; the first is the targetname of the entity whose keyvalue(s) you wish to alter, and the second (optional) value is the new value for "target" that you may wish to assign to the targeted entity. If two values are used, then they should be separated by a comma.
Syntax: targeted_entity_name,new_target_value.

targetname

The name of the specific target_change.

[various]

Virtually any other keyvalue may be assigned, assuming the targeted entity can use it.

Spawnflags

[various]

Virtually any spawnflag value may be assigned, assuming the targeted entity can use them.



Target_changelevel

The **Lazarus** target_changelevel has a few significant capabilities beyond that of the standard Quake2 target_changelevel point entity. The first is the addition of a **CLEAR_INVENTORY** flag, which enables you to reset the player inventory and health to game start conditions (kinda like Quake1 when you finish an episode).

The second change is the addition of a **NOGUN** flag. When set, when the next map or demo is loaded, the cvars **cl_gun** and **crosshair** will be set to 0. This is useful for creating demo (.dm2) intermissions which function to move your story along, without the gun and crosshair showing. When that map or demo is exited, these two cvars will have their previous settings restored.

Third, you can specify the skill level for the next map by use of the **EASY**, **NORMAL**, **HARD**, or **NIGHTMARE** spawnflags. What's this useful for? Remember the opening hub map in Quake? Well here ya go. This feature is useful for cases where there is only one choice of the next map to play. If you'd rather build a hub in which there are multiple map choices **and** multiple skill level choices, use **target_skill** to set the skill level rather than target_changelevel. Otherwise you'll need 3-4 target_changelevels for each map transition. Skill value set with target_skill overrides the target_changelevel setting (if any).

The fourth allows very seamless level changes. The **LANDMARK** flag causes the player to spawn in a position relative to the player start that is equal to what his position was relative to the changelevel in the previous level. This means contrived choke points for level changes are no longer necessary. The player can change levels while inching along the wall of a wide room in one level, and spawn right next to the wall in the next level, in a manner similar to a Half-Life level change. What's more, player facing angle, view pitch angle, and velocity are inherited. This means that it's possible to place a level change in the middle of a large warehouse, and the player can change levels in mid-leap from crate to crate - and when he spawns in the next map, he'll land on the crate he intended to jump to.

To give more flexibility to this feature, LANDMARK changelevels now will recognize **angle**. This is useful if you have 2 maps in a series that aren't oriented the same way. The value of angle will increment player facing angle and will reorient offset and velocity to match.

LANDMARK changelevels also support the use of the **trigger_transition**. When used, the trigger_transition will allow other entities to change maps when the player does. The target_changelevel's target value is used to reorient all entities within the trigger_transition field. To use a trigger_transition with a target_changelevel, **LANDMARK must** be set.

There are 2 changes made to **every** target_changelevel that doesn't use *unit change **map** key values, and this potentially affects **every** map run under **Lazarus**:

- When the player spawns in the next level, the player's gunframe is the frame used when the player triggered the changelevel in the previous map. This means that there's no more of that goofy "pulling out the gun" effect every time the player changes maps.
- Primarily to support the use of the **trigger_transition**, the player will now not suffer any loss of health below 10 health points for the first 3 seconds after a new map loads. (This does not affect armor). This is done to give the player a chance to find health, finish off his enemy, and/or take cover - rather than possibly leave him with a worthless autosaved game.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the number of degrees to increment player facing angle when spawning in the next map. Position offset and velocity is also rotated. Only relevant when **LANDMARK** is set. Default=0. Setting angle allows maps in a series to be oriented differently from each other to still use LANDMARK.

map

Name of the next map, demo, cinematic, or image to load.

targetname

Name of the specific target_changelevel.

Spawnflags

CLEAR_INVENTORY Spawnflag (=1)

Changelevel will cause all weapons, ammo and items to be stripped from the player on level change. Player health is also reset to 100.

LANDMARK Spawnflag (=2)

Changelevel is used as a landmark to determine spawning position in the next level. For example, if the player is 128 units away on the X-axis from the origin of the target_changelevel when the changelevel is triggered, he will spawn 128 units away on the X-axis from the origin of the info_player_start in the next map. This naturally means that the placement of a LANDMARK target_changelevel is critical. Care should be taken that the placement of both the changelevel and the player start is exact, and that there is no difference in the relevant architecture from one map to the other. Failure to do this may result in the player getting stuck in a wall when he spawns in the next map.

Also, at the time the changelevel is triggered, player facing direction, view pitch angle, and velocity are saved. When the player spawns in the next map, he will inherit those values. If the 2 maps in a series are not oriented identically, this can be compensated for by setting an **angle** value equal to the number of degrees that you want to increment the player facing angle by.

This spawnflag **must** be set in order to use a **trigger_transition** in conjunction with the target_changelevel.

NOGUN Spawnflag (=4)

The target_changelevel will force the cvars **cl_gun** and **crosshair** to be set to 0 for the next map or demo loaded. Upon exiting this map or demo, the values of these cvars will be restored to their previous settings.

EASY Spawnflag (=8)

Skill level will be set to 0 for the next map.

NORMAL Spawnflag (=16)

Skill level will be set to 1 for the next map.

HARD Spawnflag (=32)

Skill level will be set to 2 for the next map.

NIGHTMARE Spawnflag (=64)

Skill level will be set to 3 for the next map.

NOT_IN_EASY Spawnflag (=256)

The target_changelevel will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_changelevel will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_changelevel will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_changelevel will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Target_crosslevel_trigger

The **Lazarus** target_crosslevel_trigger differs from the standard Quake2 target_crosslevel_trigger point entity in that it offers much of the functionality of the trigger_relay as well as the ability to set crosslevel target flags.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

delay

Specifies the delay in seconds before the target_crosslevel_trigger will fire after being triggered.
Default=0.

killtarget

Targetname of the entity to be removed from the map when the target_crosslevel_trigger fires.

message

Specifies the character string to print to the screen when the target_crosslevel_trigger fires.

target

Targetname of the entity to be triggered when the target_crosslevel_trigger fires.

targetname

Name of the specific target_crosslevel_trigger.

Spawnflags

TRIGGER_1 Spawnflag (=1)

Sets the first of 8 possible flags read on map load for the purpose of triggering a target_crosslevel_target.

TRIGGER_2 Spawnflag (=2)

Sets the second of 8 possible flags read on map load for the purpose of triggering a target_crosslevel_target.

TRIGGER_3 Spawnflag (=4)

Sets the third of 8 possible flags read on map load for the purpose of triggering a target_crosslevel_target.

TRIGGER_4 Spawnflag (=8)

Sets the fourth of 8 possible flags read on map load for the purpose of triggering a target_crosslevel_target.

TRIGGER_5 Spawnflag (=16)

Sets the fifth of 8 possible flags read on map load for the purpose of triggering a target_crosslevel_target.

TRIGGER_6 Spawnflag (=32)

Sets the sixth of 8 possible flags read on map load for the purpose of triggering a target_crosslevel_target.

TRIGGER_7 Spawnflag (=64)

Sets the seventh of 8 possible flags read on map load for the purpose of triggering a target_crosslevel_target.

TRIGGER_8 Spawnflag (=128)

Sets the eighth of 8 possible flags read on map load for the purpose of triggering a target_crosslevel_target.

NOT_IN_EASY Spawnflag (=256)

The target_crosslevel_trigger will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_crosslevel_trigger will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_crosslevel_trigger will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_crosslevel_trigger will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Target_effect

Target_effect is similar to a target_temp_entity, but allows you to produce **any** of the standard Quake2 particle effects. The effect to produce is specified by the **style** value, and different styles take different parameters. Several of the explosion effects are identical but have been included for completeness. See effects.bsp in the **examples**.

Spawnflags

LOOPED_ON

If set, the target_effect will be on when the map loads, and repeat every **wait** seconds.

LOOPED_OFF

If set, the target_effect will be off when the map loads. Once triggered, the target_effect will repeat every **wait** seconds.

IF_MOVING

If set and the target_effect uses the **movewith** key, the effect will **only** be displayed when the movewith target is in motion.

Key/Value Pairs

movewith

Targetname of the entity to move with

targetname

Name of the target_effect. This entity must be triggered (normally by a func_timer) to work, unless LOOPED_ON is set.

style

Specifies the type of effect to produce. Use the number value in the list below to select the desired effect.

0 TE_GUNSHOT	26 TE_GREENBLOOD
1 TE_BLOOD	28 TE_PLASMA_EXPLOSION
2 TE_BLASTER	29 TE_TUNNEL_SPARKS
3 TE_RAILTRAIL	30 TE_BLASTER2
4 TE_SHOTGUN	33 TE_LIGHTNING
5 TE_EXPLOSION1	34 TE_DEBUGTRAIL
6 TE_EXPLOSION2	35 TE_PLAIN_EXPLOSION
7 TE_ROCKET_EXPLOSION	36 TE_FLASHLIGHT
8 TE_GRENADE_EXPLOSION	38 TE_HEATBEAM
9 TE_SPARKS	39 TE_MONSTER_HEATBEAM
10 TE_SPLASH	40 TE_STEAM
11 TE_BUBBLETRAIL	41 TE_BUBBLETRAIL2
12 TE_SCREEN_SPARKS	42 TE_MOREBLOOD
13 TE_SHIELD_SPARKS	43 TE_HEATBEAM_SPARKS
14 TE_BULLET_SPARKS	44 TE_HEATBEAM_STEAM
15 TE_LASER_SPARKS	45 TE_CHAINFIST_SMOKE
16 TE_PARASITE_ATTACK	46 TE_ELECTRIC_SPARKS
17 TE_ROCKET_EXPLOSION_WATER	47 TE_TRACKER_EXPLOSION
18 TE_GRENADE_EXPLOSION_WATER	48 TE_TELEPORT_EFFECT
19 TE_MEDIC_CABLE_ATTACK	49 TE_DBALL_GOAL
20 TE_BFG_EXPLOSION	50 TE_WIDOWBEAMOUT
21 TE_BFG_BIGEXPLOSION	51 TE_NUKEBLAST
22 TE_BOSSTPORT	52 TE_WIDOWSPLASH
23 TE_BFG_LASER	53 TE_EXPLOSION1_BIG
24 TE_GRAPPLE_CABLE	54 TE_EXPLOSION1_NP
25 TE_WELDING_SPARKS	55 TE_FLECHETTE

target

Entity to aim the effect at. This value is **only** used for TE_RAILTRAIL, TE_BUBBLETRAIL, TE_PARASITE_ATTACK, TE_MEDIC_CABLE_ATTACK, TE_BFG_LASER, TE_GRAPPLE_CABLE, TE_LIGHTNING, TE_DEBUGTRAIL, TE_HEATBEAM, TE_MONSTER_HEATBEAM, and TE_BUBBLETRAIL2. A target **must** be specified for these effects. Failure to specify a target will result in a runtime error

message, and the target_effect will be destroyed. If the target does not exist at the time the target_effect is triggered, target_effect does nothing.

angles

Used for most of the target_effect styles other than those that use a target. The pitch and yaw angles specify the movement direction for the effect.

count

Number of particles to produce each time target_effect is triggered. This value is only used by TE_SPLASH, TE_LASER_SPARKS, TE_WELDING_SPARKS, and TE_STEAM.

sounds

Color used by the target_effect. Unfortunately this gets to be a bit confusing, as *sounds* has a different meaning depending on which style is used. For TE_SPLASH, *sounds* is identical to the definition for a target_splash:

- 1) sparks
- 2) blue water
- 3) brown water
- 4) slime
- 5) lava
- 6) blood

For TE_LASER_SPARKS, TE_WELDING_SPARKS, and TE_STEAM, *sounds* is the index in the standard Quake2 palette to use. A few examples:

- 6-9 - varying whites (darker to brighter)
- 80 - brown water
- 176 - blue water
- 208 - slime
- 224 - sparks
- 232 - blood

wait

If not looped, used **only** by TE_STEAM. If non-zero, this is the duration of the steam effect (in seconds), and will override triggering events by a func_timer. For looped target_effects, *wait* specifies the cycle time for the effect.

speed

Used **only** by TE_STEAM and TE_TUNNEL_SPARKS. Speed of the particles, in units/second. Default value = 75. The width of the base is proportional to the speed.

Notes:

- Many effects have sounds associated with them. Disabling those sounds is not an option, unless you use "silent" .wav files using the same filename in your specific game folder.
- Some effects seem to suck the life out of other effects running at the same time. This can be seen most easily with the TE_BOSSTPORT - when this is turned on, other effects will just shut down.
- TE_WIDOW_BEAMOUT is a cool looking effect but has an apparent bug in the executable. If used alone in a map, TE_WIDOW_BEAMOUT gives a nice spherical ball of particles. However, in the **example map** this effect is a bit unpredictable.
- Several effects supported by Lazarus require external models or make use of sounds that (at present) are not distributed with Lazarus:

Effect	Model used
TE_LIGHTNING	Rogue's tesla lightning bolt (models/proj/lightning/tris.md2) sounds/weapons/tesla.wav
TE_EXPLOSION1_BIG	Rogue's models/objects/r_explode2/tris.md2
TE_HEATBEAM TE_MONSTER_HEATBEAM	Rogue's models/proj/beam/tris.md2
TE_GRAPPLE_CABLE	CTF's models/ctf/segment/tris.md2
TE_TRACKER_EXPLOSION	Rogue's sounds/weapons/disrupthit.wav

Lazarus Main Page

Target_explosion

The **Lazarus** target_explosion is identical to the standard Quake2 target_explosion point entity, with the addition of **movewith** support, which allows it to move with its parent entity, and the **count** key, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

When non-zero, specifies the number of times the target_explosion will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

delay

Specifies the delay in seconds before the target_explosion will fire after being triggered. Default=0.

dmg

Specifies the amount of damage hit points the target_explosion will generate at its origin. Default=0.

killtarget

Targetname of the entity to be removed from the map when the target_explosion is triggered.

movewith

Targetname of the parent entity the target_explosion is to **movewith**.

targetname

Name of the specific target_explosion.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_explosion will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_explosion will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_explosion will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_explosion will be inhibited and not appear when deathmatch=1.

Target_fade

The **Lazarus** target_fade point entity can be used to fade the user's screen to a specified **color** and **alpha** (obscuration) value. This might be useful when used in conjunction with explosions and/or the dreaded bottomless pit scenario. You may supply 3 time values (**fadein**, **holdtime**, and **fadeout**) to control the rate at which the fade occurs and how long the fade persists.

Software rendering presents a bit of a problem for fading the display. Namely, **all** screen elements, including menus, are affected by the fade. If precautions were not taken it might be possible for the player to trigger a solid black target_fade and die before the target_fade effect was removed. In that case the player would be left with a solid black display and no visible menu items or controls. For this reason Lazarus automatically removes all target_fade effects when the player dies if software rendering is in use. This same scenario poses no problems for 3D cards, as menus are visible regardless of what target_fade parameters are in effect.

Key/value pairs

alpha

Alpha value (obscuration) of the fade, 0-1. 1 gives a solid color, 0 results in no effect at all. Default=0.

color

Red, green, and blue values (0-1) specifying the color of the effect. Default=0 0 0 (black).

count

When non-zero, specifies the number of times the target_fade will be called before being auto-killtargeted (see [this page](#) for details). The code will automatically change the value of count to 1 when **holdtime**<0. Default=0.

fadein

Time in seconds from activation until the specified **alpha** value is reached. Default=0.

fadeout

Time in seconds from the conclusion of **holdtime** until the display returns to normal. Ignored when **holdtime**<0. Default=0.

holdtime

Time in seconds after the conclusion of **fadein** to hold the effect at a constant **alpha**. A negative value makes the effect persistent, and the target_fade cannot be triggered again. Default=0.

targetname

Name of the specific target_fade.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_fade will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_fade will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_fade will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_fade will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Target_failure

The target_failure point entity is sort of a special-purpose **target_text** which is designed to inform the player when he commits any sort of mission-critical mistake. When triggered, a **message** will appear, the screen will fade to black, and the game will end just as if the player died - but note that he doesn't have to die for this to happen.

To use this entity logically, the triggering mechanism should be some sort of mission-ending mistake, like killing the wrong guy or destroying something necessary.

As outlined in the **redistribution** doc, if the **NO_BACKGROUND** spawnflag is **not** set, this entity will require the image file **pics/textdisplay.pcx**.

You might find the **formatting notes** on the target_text page helpful, as they apply to this entity as well.

Key/value pairs

message

Specifies the string to be displayed when the target_failure is triggered. If the **FILE** spawnflag is set, then this key specifies the filename of the external file to retrieve text from.

noise

Specifies the .wav file to be played when the target_failure is triggered. Syntax: [path]/[file].wav.

targetname

Name of the specific target_failure.

Spawnflags

FILE Spawnflag (=1)

The text to be displayed will be retrieved from an external file, rather than use the text string specified by **message**, which would instead specify the filename of that external file. The file should be located in the /MAPS/ directory or in a subdirectory of /MAPS/.

NO_BACKGROUND Spawnflag (=2)

The text will be displayed without a background or plaque.

NOT_IN_EASY Spawnflag (=256)

The target_failure will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_failure will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_failure will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_failure will be inhibited and not appear when deathmatch=1.



Target_fog

The **Lazarus** target_fog point entity provides one method of introducing fog effects in your map. Unlike **trigger_fog**, target_fog causes a global (map-wide) fog effect to be rendered when it is triggered. If you are unfamiliar with how **Lazarus** fog works, then get the whole scoop on the **Fog Effects** page.

Target_fog will not override what the player sees when he is inside a **trigger_fog** field.

Fog can be made directional (the player sees denser fog looking in one direction than he would looking in the opposite direction). This is done by giving **fog_density** and **density** different values, and setting view direction with **angle** or **angles**.

Key/value pairs

angle

Specifies the player facing direction on the XY plane where he will see the fog density value as specified by **fog_density**. Default=0. Only relevant for "directional" fog; ignored if **density**=0.

angles

Specifies the player facing direction in 3 dimensions, defined by pitch and yaw (roll is ignored), where he will see the fog density value as specified by **fog_density**. Default=0 0 0. Only relevant for "directional" fog; ignored if **density**=0.

count

When non-zero, specifies the number of times the target_fog will be called before being auto-killed (see [this page](#) for details). Default=0.

delay

Specifies the time in seconds that the target_fog will ramp fog density up/down. Density ramps from the player's currently rendered fog level to the fog density specified by the target_fog. Default=0.

density

When non-zero, specifies "directional" fog. Non-zero values specify the level of fog density viewed by the player when his view is directly **opposite** the direction specified by **angle/angles**. For best results, use values of <100. Default=0. Ignored when **fog_model**=0.

fog_color

Specifies relative RGB color components of the fog effect; valid range for each component is 0.0-1.0. Default=0.5 0.5 0.5.

fog_density

Specifies the level of fog density as viewed by the player. If **density** is non-zero, the fog is "directional", and fog_density will specify the fog density of the player when his view is aligned with the direction specified by **angle/angles**. For best results, use values of <100. Default=20. Ignored when **fog_model**=0.

fog_far

Specifies the distance in map units from the player's viewpoint where visibility will be completely obscured by fog. See also **fog_near**. Default=1024. Ignored if **fog_model**>0.

fog_model

Specifies the method used to calculate how fog density increases with distance. Default=1. Note: Linear fog cannot be "directional". Choices are:

- 0: Linear
- 1: Exponential
- 2: Fast exponential

fog_near

Specifies the distance in map units from the player's viewpoint where visibility will first begin to be obscured by fog. See also **fog_far**. Default=20. Ignored if **fog_model**>0.

targetname

Name of the specific target_fog.

Spawnflags**START_ON Spawnflag (=1)**

The fog parameters for the target_fog will be active when the map loads. Of course only one target_fog per map should use this spawnflag.

TOGGLE Spawnflag (=2)

The target_fog can be toggled on and off. Fog density levels will ramp up/down as specified by **delay**.

TURN_OFF Spawnflag (=4)

The target_fog functions as a fog effect canceller. All keyvalues except for **delay** and **targetname** will be ignored.

NOT_IN_EASY Spawnflag (=256)

The target_fog will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_fog will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

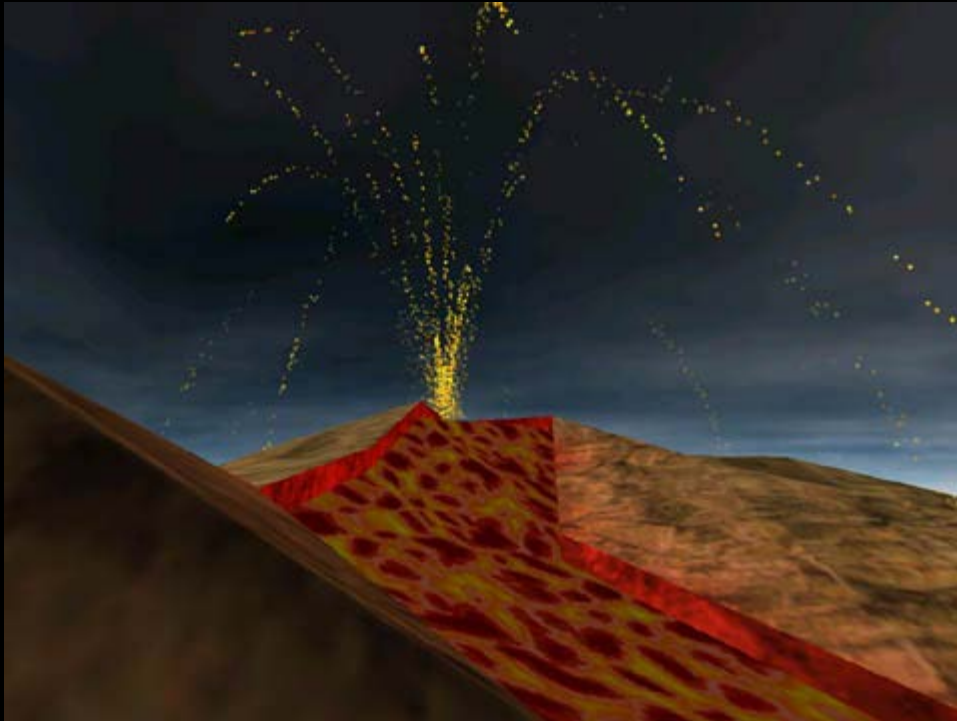
The target_fog will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_fog will be inhibited and not appear when deathmatch=1.

Target_fountain

The **Lazarus** target_fountain allows you to generate a steady stream of models, thrown from a point source in a semi-random direction. It is functionally very similar to **target_precipitation**. The key differences from target_precipitation are that there are no predefined styles (rain, leaves, etc.) and the model placement and direction are determined differently. While target_precipitation spawns models at random locations and all models move in the same direction, models generated by target_fountain all start at the same location but then move in random directions.



Usermodel=point.sp2 (invisible sprite), Effects=GREENGIB

How it works: Every 0.1 second frame, target_fountain spawns a specified number of models at the target_fountain origin. Direction of travel for the models is determined by the placement of a bounding box described by **bleft** and **tright** - a random point within that box is selected, and the vector from the origin to that point is the direction the model will initially move in. The number of models generated per second is specified with the **count** and **random** values. The number of models spawned in any frame will be $0.1 * (\text{count} \pm \text{random})$, with the remainder after truncating to an integer carried over to the next frame. Spawned models are removed when they touch a solid object, or at an optional **fadeout** time after touching a solid.

NOTE: All of the warnings concerning SZ_GetSpace: Overflow errors with target_precipitation apply to target_fountain as well. Be sure to read up on potential problems you might face in the **target_precipitation** documentation.

Key/value pairs

attenuation

Controls the rebound of models after striking a solid. This value is ignored if **gravity** is 0, or if the **BOUNCE** spawnflag is not set. 0=no bounce, 0.9 or so is a SuperBall. 1.0 and higher are physically impossible but still allowed. As a reference point, metallic gibs and grenades use a value of 0.5. Default=0.

bleft

Specifies bottom-left (minX/minY/minZ) bounding box coordinates of the area within which a point defining the initial movement direction will be located. See also **tright**. Default=(-32 -32 64).

count

Specifies the base number of models spawned each second. The number of entities spawned in any 0.1 second frame will be in the range $0.1 * (\text{count} \pm \text{random})$. Default=1.

effects

Effects may be combined by adding the values.

- 1: ROTATE Rotate like a weapon
- 2: GIB
- 8: BLASTER - Yellowish orange glow plus particles
- 16: ROCKET - Rocket trail
- 32: GRENADE - Grenade trail
- 64: HYPERBLASTER - BLASTER w/o the particles
- 128: BFG - Big green ball
- 256: COLOR_SHELL
- 512: POWERSCREEN - Green power shield
- 16384: FLIES - Ewwwww
- 32768: QUAD - Blue shell
- 65536: PENT - Red shell
- 131072: TELEPORTER - Teleporter particles
- 262144: FLAG1 - Red glow
- 524288: FLAG2 - Blue glow
- 1048576: IONRIPPER
- 2097152: GREENGIB
- 4194304: BLUE_HB - Blue hyperblaster glow
- 8388608: SPINNING_LIGHTS - Red spinning lights
- 16777216: PLASMA
- 33554432: TRAP
- 67108864: TRACKER
- 134217728: DOUBLE - Yellow shell
- 268435456: SPHERETRANS - Transparent
- 536870912: TAGTRAIL
- 1073741824: HALF_DAMAGE
- 2147483648: TRACKER_TRAIL

fadeout

Fadeout specifies the time, in seconds, that a model will persist once it touches a solid object before fading away. Keep in mind that the larger fadeout is, the more entities will exist at any given time. Default=0.

gravity

Specifies the ratio of normal gravity that the model will be attracted to the ground with. 1=normal gravity, 0=move at a constant **speed** value. If gravity is non-zero **and** **attenuation** is non-zero, the model will bounce when it strikes a solid brush, monster, or player. Default=0.

mass

Weight of the user-defined model, used in determining damage to monsters or players struck by the model. Damage inflicted is proportional to the mass times the impact velocity. This value is ignored if **gravity** is 0. Default = 0.

mass2

Number of splash particles produced when the model strikes a solid object or liquid brush. This value has no effect if **SPLASH** is not set. Default=8.

pitch_speed

Specifies the maximum angular velocity around the Y axis of the model, in degrees/second. When spawned, the model will be given a constant angular velocity in the range $\pm \text{pitch_speed}$. Default=0.

random

Modifier to **count**. The number of entities spawned per second will be $\text{count} \pm \text{random}$. Default=0.

renderfx

Effects may be combined by adding the values.

- 1: MINLIGHT Never completely dark
- 2: VIEWERMODEL
- 4: WEAPONMODEL
- 8: FULLBRIGHT
- 16: DEPTH HACK
- 32: TRANSLUCENT Transparent
- 64: FRAMELERP
- 128: BEAM
- 512: GLOW Pulsating glow of normal Q2 pickup items
- 1024: SHELL_RED
- 2048: SHELL_GREEN
- 4096: SHELL_BLUE
- 32768: IR_VISIBLE
- 65536: SHELL_DOUBLE
- 131072: SHELL_HALF_DAMAGE White shell

roll_speed

Specifies the angular velocity around the X axis of the model, in degrees/second. When spawned, the model will be given a constant angular velocity in the range \pm roll_speed. Default=0.

sounds

Specifies the color index in the Q2 palette of splash particles. Default=183 (blue)

speed

Initial speed of spawned models. Default=300.

targetname

Name of the specific target_fountain.

tright

Specifies top-right (maxX/maxY/maxZ) bounding box coordinates of the area within which a point defining the initial movement direction will be located. See also **bleft**. Default=(32 32 128).

usermodel

Specifies the name of the model to spawn. The filename should be relative to <gamedir>/models for .md2 files, or <gamedir>/sprites for .sp2 files.

yaw_speed

Specifies the angular velocity around the Z axis of the model, in degrees/second. When spawned, the model will be given a constant angular velocity in the range \pm yaw_speed. Default=0.

Spawnflags

START_ON Spawnflags (=1)

If set, target_fountain will start when the first player spawns into the map. Otherwise it must be triggered to work.

SPLASH Spawnflag (=2)

If set, a splash effect will be generated when the model touches a solid or liquid brush or a solid entity. Color of the splash particles is controlled by the **sounds** value. Number of splash particles is set with **mass2**.

BOUNCE Spawnflag (=4)

If set, spawned models will bounce when they touch a solid. Bounce physics is determined by the **gravity** setting.

FIRE_ONCE Spawnflag (=8)

If set, target_fountain will produce models only once when triggered, then shut itself off.

START_FADE Spawnflag (=16)

If set and fadeout is non-zero, models will begin to fade away when first spawned, rather than waiting until they strike a solid.

NOT_IN_EASY Spawnflag (=256)

The target_fountain will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_fountain will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_fountain will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_fountain will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Target_help

The **Lazarus** target_help is identical to the standard Quake2 target_help point entity, with the addition of **count** support, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

When non-zero, specifies the number of times the target_help will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

message

Specifies the message that will appear in the F1 computer screen. Message appears in the Secondary Objective window unless **MAIN_OBJECTIVE** is set.

targetname

Name of the specific target_help.

Spawnflags

MAIN_OBJECTIVE Spawnflag (=1)

Specifies the **message** will appear in the Main Objective window rather than the Secondary Objective window.

NOT_IN_EASY Spawnflag (=256)

The target_help will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_help will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_help will be inhibited and not appear when skill=2 or greater.

[Lazarus Main Page](#)

Target_laser

The **Lazarus** target_laser differs from the standard Quake2 target_blaster point entity in that the laser can be given a pulse effect (by using the **delay** and **wait** keys), and the laser can also make use of a **SEEK_PLAYER** spawnflag which lets you target the player.

A further modification involves the use of negative **dmg** values. As you may know, negative-damage lasers will give health. But where the normal Q2 laser will cause players and monsters to groan in pain and spit out blood pixels no matter what the dmG value, negative-damage **Lazarus** lasers do no such thing. If you want a true zero-damage laser, then you'll need to use the **style** key.

What's more, you aren't limited to the 2 choices of the default laser width or the **FAT** width. You can set a **mass** value, which can override the thin/fat settings and specify widths smaller, larger, or anything in between.

As with many other **Lazarus** entities, the target_laser can also be made to **movewith** a parent entity, and includes **count** support, which allows the laser to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies an aiming direction on the XY plane. Default=0. Ignored if **target** is used, and/or **SEEK_PLAYER** is set.

angles

Specifies an aiming direction in 3 dimensions, defined by pitch and yaw (roll is ignored). Default=0 0 0. Ignored if **target** is used, and/or **SEEK_PLAYER** is set.

count

When non-zero, specifies the number of times the laser will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

delay

Specifies the amount of time, in seconds, that a laser pulse will be on. This value must be less than the value of **wait**. Default=0. Ignored if **wait**=0.

dmg

Specifies the number of damage hit points the laser will do every 0.1 seconds. Default=100. If set to a negative value, the laser will give health, but will not cause pain effects. Ignored when **style**>0.

mass

Sets the width of the laser. When 0, uses the default internal value of 4 for normal lasers, or 16 for fat lasers. When non-zero, **FAT** is ignored. Default=0.

movewith

Targetname of the parent entity the laser is to **movewith**.

style

Sets the behavior of the laser. When non-zero, **dmg** is ignored. Choices are:
0: Affects health, monsters avoid, creates sparks (normal, default laser).
1: No damage; monsters avoid; creates sparks.
2: No damage; monsters ignore; creates sparks.

3: No damage; monsters ignore; no sparks.

target

Targetname of the entity the laser will fire at. If **SEEK_PLAYER** is set, the player is a higher priority target, and therefore the entity specified by "target" will be ignored, assuming the laser has a clear shot at the player. In the event **SEEK_PLAYER** is set and the laser has no clear shot at him, it will fall back to firing at its "target".

targetname

Name of the specific target_laser.

wait

If non-zero, specifies that the laser will pulse on and off automatically. The value specified here sets the amount of time (in seconds) between pulse cycles. Default=0. Also requires the use of **delay**. (Please see the **notes** below regarding timing issues when using pulse lasers).

Spawnflags

START_ON Spawnflag (=1)

The laser will be active when the map loads.

RED Spawnflag (=2)

The laser will be red.

GREEN Spawnflag (=4)

The laser will be green.

BLUE Spawnflag (=8)

The laser will be blue.

YELLOW Spawnflag (=16)

The laser will be yellow.

ORANGE Spawnflag (=32)

The laser will be orange.

FAT Spawnflag (=64)

The laser will use the large diameter animation frames rather than the narrow diameter frames. Ignored when **mass** is non-zero.

SEEK_PLAYER Spawnflag (=128)

Specifies that the target_laser will target the first player it finds, and will consider players priority targets over entities specified by **target**. The laser will fall back to firing at its **target** (if any), if it has no clear shot at a player.

NOT_IN_EASY Spawnflag (=256)

The target_laser will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_laser will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_laser will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_laser will be inhibited and not appear when deathmatch=1.

Notes

The fact that you can now specify laser pulse rates and duration with the **wait** and **delay** keys may suggest a number of timed sequenced events that may be triggered with them. Sadly, it's really pretty meaningless to try to put too fine a resolution on these laser timings. Like all entities other than the player, the target_laser only "thinks" every 0.1 seconds... and that's only if the game isn't too bogged down with other stuff going on at the same time. For example, if you set delay=0.4, wait=0.5, then you may well get a continuous pulse at one time, and at another you might get a 0.3 second burst with the next burst occurring 0.3 seconds later. Bottom line: do **not** rely on laser timings for any sort of scripted sequence.

[Lazarus Main Page](#)

Target_Lightramp

Lazarus adds the capability to use a `target_lightramp` to create your own **lightstyles**. Lightstyles as provided in the standard Quake2 code allow for only a limited number of choices, which may or may not be appropriate for the effect you want to create. Note that all switched lights and lightstyles are disabled in multiplayer.

The normal use of the **target_lightramp** is to make lights fade in and out. This is done by giving a starting light level (represented by a lowercase alphabet character) and an ending light level (represented by another character). These two characters are used as the value for the lightramp's **message** key. The lowest possible light level is "a", normal light is "m", and the brightest light is "z". So, if you wanted to make a light start from darkness and brighten to normal level, you would make the value of **message** = **am**. The amount of time it would take this fade effect to take place is set with the **speed** key.

To make use of the **Lazarus**-added ability to create your own lightstyle, set the **CUSTOM** spawnflag. When this is set, the **message** key can now take as many characters as you want. Each character represents a different light level that will be used every **0.1 seconds**. If you want a particular light level to be held for a longer period of time, then repeat the character as many times as needed. Naturally, **CUSTOM** lightramps ignore the **speed** value, if any.

For both **CUSTOM** and non-**CUSTOM** lightramps, the **TOGGLE** spawnflag causes the lightramp to reverse its pattern for every other trigger. In other words, if a lightramp's **message** = **abc**, then the second time it's triggered it would run the pattern **cba**.

To further facilitate user-defined lightstyles, a **LOOP** spawnflag is also available. When set, once the lightramp is triggered it would repeat the **message** string (read left to right only) until it is triggered again to turn it off. If both **LOOP** and **TOGGLE** are set, the pattern is read normally, then reversed, and this would continue until triggered again. A **LOOP** lightramp will complete its message when turned off rather than stopping in mid-pattern.

It's significant to note that normal Quake2 does not allow switchable light entities to use any lightstyles. With this modified lightramp, it's now possible to turn a lightsyle light on and off, since the style is defined in the lightramp and not in the light. (See the example map, which uses a **target_rotation** and **trigger_relays** to make this happen). This also allows the use of flashing light effects that occur only occasionally, where in normal Quake2 this could only be done with multiple switched lights of varying brightness levels... and then this would be limited to only four switched lights in a single area. More lights than that and you'd get a "Face with too many lightstyles" radiosity compiling error, and the area would be lit oddly in the game. The **Lazarus** lightramp lets you get around that, since only a single style light entity would be needed.

For informational purposes, these are the lightsyle codes behind the options the regular game offers for light entities, so you can get an idea of what can be used as the lightramp's message:

1. Flicker #1 : **mmnmommommnonmmonqnmmo**
2. Slow Strong Pulse : **abcdefghijklmnpqrstuvwxyzyxwvutsrqponmlkjihgfedcba**
3. Candle #1 : **mmmmaaaaaammmmmaaaaaabcedefgabcdefg**
4. Fast Strobe : **mamamamamama**
5. Gentle Pulse : **jklmnopqrstuvwxyzyxwvutsrqponmlkj**
6. Flicker #2 : **nmonqnmommomomno**
7. Candle #2 : **mmmaaaabcedfgmmmmaaaaammmaamm**
8. Candle #3 : **mmmaaaammmaammmaabcedfaaaammmmaabcedfmmmaaaa**
9. Slow Strobe : **aaaaaaaaazzzzzzz**
10. Flourescent Flicker : **mmamammmmmammamamaaaamamma**
11. Slow Pulse, No Black : **abcdefghijklmnpqrrqponmlkjihgfedcba**

Key/Value pairs

count

When non-zero, specifies the number of times the `target_lightramp` will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

message

Sets the light levels of the **targeted** lights. Light levels are defined by characters **a-z**. Normally, only 2 characters can be used, but if the **CUSTOM** spawnflag is set, many characters can be used in a string to define a custom **lightstyle pattern**.

speed

Amount of time in seconds from the starting **message** value to the ending value. If the **CUSTOM** spawnflag is set, speed is ignored.

target

Targetname of the light entity (or entities) to target.

targetname

Name of the specific target_lightramp. Lightramps must be triggered in order to function.

Spawnflags**TOGGLE (Spawnflags=1)**

Allows the lightramp to be triggered multiple times, with every other firing resulting in the reverse effect (as an example, lights that fade in on the first trigger will fade out on the second trigger).

CUSTOM (Spawnflags=2)

Enables the ability to create custom lightstyles by allowing many characters to be used as the **message** string. When set, **speed** is ignored.

LOOP (Spawnflags=4)

Once triggered, the lightramp will continue its **message** pattern as read left to right repeatedly until triggered again. If the **TOGGLE** spawnflag is also set, then the **message** string is read left to right, the right to left, also repeatedly.

[Lazarus Main Page](#)

Target_lightswitch

Use target_lightswitch to toggle all lights in a level on/off. This feature works fairly well, with these exceptions:

1. It should **not** be used where sky brushes are visible. The appearance of the sky brushes does not change when the lights are dimmed - they will no longer give off light, but they will appear much too bright.
2. Transparent brushes (windows, for example) will keep the same light level they had with normal lighting.
3. Rotating entities will be completely black, even when lit by [item_flashlight](#).

Only one target_lightswitch per map is allowed.

Key/Value pairs

message

Minimum light value, on a scale of "a" to "m". Default = "a".

count

When non-zero, specifies the number of times the target_lightswitch will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

[Lazarus Main Page](#)

Target_locator

The target_locator is a random entity locator. Use it to spawn an entity from a random selection from an array of path_corners in the map. This feature allows you to add quite a bit of replay value to your map with a minimum of fuss. The target_locator **target** key is the targetname of the entity you want to move. This can be a point entity (health, items, keys, etc.) OR a brush model (func_pushable, for example). Use the **pathtarget** key to specify the first in a series of path_corners. In the case of brush models, the path_corner locations specify the geometric center of the model. Brush models will be lit as though placed in the location used in the .map file. The entity will "droptofloor" when placed at the path_corner location. **NOTE:** You should not attempt to re-use path_corners for multiple target_locators. Since the selection of path_corners is completely random, doing this may cause multiple entities to occupy the same spot. This is a bad thing (TM). Exception: One target_locator can be used to target multiple entities with the same targetname, and the associated path_corners will not be re-used... every entity with the same targetname will occupy a unique path_corner. If you foul up and use more entities (N) with the same targetname than you have path_corners (M), the code will stop re-locating after the Mth entity.

Key/Value pairs

target

Name of the entity or entities to move.

pathtarget

Name of the first path_corner in a path_corner sequence to choose a location from. If the target is a point entity (rather than a brush model), it will inherit the angles values of the path_corner.

[Lazarus Main Page](#)

Target_lock

Target_lock_digit

Target_lock_code

Target_lock_clue

The combination lock uses several different entities dependent on your objective: All combination locks use one `target_lock` + between one and eight `target_lock_digit`'s. Each `target_lock_digit` displays one digit from the lock combination. The relationship between `target_lock` and `target_lock_digit` is very similar to `target_string` and `target_character`. The `target_lock` and each `target_lock_digit` must have the same "team" value. `Target_lock` should have a `targetname` and a `target`. In practice, a `trigger_multiple` or `func_button` will generally be used to target the `target_lock`. If the correct combination has been entered with the `target_lock_digits`, `target_lock` fires its `target`. If the correct combination has **not** been entered, `target_lock` triggers its `pathtarget` (if used) or plays the annoying `talk1.wav` we all love and displays its message, if one is specified. The initial display of the `target_lock_digits` is specified with the `key_message` field of `target_lock`. If unspecified, the digits will initially display all 0's. At startup, the game generates a random number and uses that number as the combination. This gives the `target_lock` a bit of replay value.

`Target_lock_digit` is a brush model that should use the `test/num_0` texture, or similar animated texture if you don't care for this one. Along with the "team" value, each `target_lock_digit` should have a unique "count" value. The count value specifies the position of the digit within the combination string. 1 is the left-most digit. You may use as many `target_lock_digits` as you wish, but more than 6 or so is rather pointless. The display is incremented by looking at the `target_lock_digit` and pressing `+use`.

The combination to a `target_lock` may be revealed to the player in several ways. The simplest method is the use of a `target_lock_code`. When triggered, the `target_lock_code` reveals the combination of the associated `target_lock`. The `target` of `target_lock_code` should be the `targetname` of the `target_lock`.

Alternatively, you can use one or more `target_lock_clue`'s. A `target_lock_clue` reveals one or more digits from the `target_lock` combination. The digits that will **not** be revealed are represented by a ? in the `target_lock_clue` message string. For example, if the message string is "X?X?", then the 1st and 3rd digits will be revealed. If the `target_lock_clue` is teamed with `target_characters`, those `target_characters` will be used to display the revealed digits of the lock combination. Initially each of the `target_characters` will scroll continuously.

Target_lock key/value pairs

targetname	Name of the lock.
team	Unique name shared by associated <code>target_lock_digits</code>
target	Name of the entity to trigger if the correct combination has been entered
pathtarget	Name of the entity to trigger if the correct combination has not been entered. If blank, the <code>target_lock</code> plays <code>talk1.wav</code>
message	Text to display if the <code>target_lock</code> is triggered with the incorrect combination.
key_message	Initial combination displayed by <code>target_lock_digits</code> . If blank, 0's are used.

Target_lock_digit key/value pairs

team	Unique name shared with <code>target_lock</code>
count	Position of this <code>target_lock_digit</code> within the combination number. 1 = 1's place, 2 = 10's place, etc.

Spawnflags

CROSSLEVEL (=1)

If you set the crosslevel spawnflag for **both** target_lock and target_lock_code, these entities may be placed in different maps **within the same unit**. If this spawnflag is set, the target_lock initially generates a random combination and stores a copy of that combination in global game information that is available to other maps (again, **within the same unit only**). The target_lock_code with the crosslevel spawnflag then reports that combination. The target_lock will not change its combination unless a) the player starts a map with the map command, or leaves the current unit (by triggering a target_changelevel whose map field begins with *). To use a target_lock with the crosslevel feature, the map with the target_lock in it **must** be played before the map with the target_lock_code. Otherwise, when the player triggers the target_lock_code he'll receive the message "Lock has not been properly initialized." You may have as many target_locks in a map as you want, but any target_locks in the same unit that use the crosslevel spawnflag will **all** have the same combination.

HUD (=2)

(Removed from Tremor for Lazarus. May be restored if there's interest) If the HUD spawnflag is set, the known digits of the target_lock combination will be shown in the HUD. Using the HUD, you could use one or more target_lock_clues to reveal the combination **without** using the accompanying target_characters.

Lazarus Main Page

Target_lock

Target_lock_digit

Target_lock_code

Target_lock_clue

The combination lock uses several different entities dependent on your objective: All combination locks use one `target_lock` + between one and eight `target_lock_digit`'s. Each `target_lock_digit` displays one digit from the lock combination. The relationship between `target_lock` and `target_lock_digit` is very similar to `target_string` and `target_character`. The `target_lock` and each `target_lock_digit` must have the same "team" value. `Target_lock` should have a `targetname` and a `target`. In practice, a `trigger_multiple` or `func_button` will generally be used to target the `target_lock`. If the correct combination has been entered with the `target_lock_digits`, `target_lock` fires its `target`. If the correct combination has **not** been entered, `target_lock` triggers its `pathtarget` (if used) or plays the annoying `talk1.wav` we all love and displays its message, if one is specified. The initial display of the `target_lock_digits` is specified with the `key_message` field of `target_lock`. If unspecified, the digits will initially display all 0's. At startup, the game generates a random number and uses that number as the combination. This gives the `target_lock` a bit of replay value.

`Target_lock_digit` is a brush model that should use the `test/num_0` texture, or similar animated texture if you don't care for this one. Along with the "team" value, each `target_lock_digit` should have a unique "count" value. The count value specifies the position of the digit within the combination string. 1 is the left-most digit. You may use as many `target_lock_digits` as you wish, but more than 6 or so is rather pointless. The display is incremented by looking at the `target_lock_digit` and pressing `+use`.

The combination to a `target_lock` may be revealed to the player in several ways. The simplest method is the use of a `target_lock_code`. When triggered, the `target_lock_code` reveals the combination of the associated `target_lock`. The `target` of `target_lock_code` should be the `targetname` of the `target_lock`.

Alternatively, you can use one or more `target_lock_clue`'s. A `target_lock_clue` reveals one or more digits from the `target_lock` combination. The digits that will **not** be revealed are represented by a ? in the `target_lock_clue` message string. For example, if the message string is "X?X?", then the 1st and 3rd digits will be revealed. If the `target_lock_clue` is teamed with `target_characters`, those `target_characters` will be used to display the revealed digits of the lock combination. Initially each of the `target_characters` will scroll continuously.

Target_lock key/value pairs

targetname	Name of the lock.
team	Unique name shared by associated <code>target_lock_digits</code>
target	Name of the entity to trigger if the correct combination has been entered
pathtarget	Name of the entity to trigger if the correct combination has not been entered. If blank, the <code>target_lock</code> plays <code>talk1.wav</code>
message	Text to display if the <code>target_lock</code> is triggered with the incorrect combination.
key_message	Initial combination displayed by <code>target_lock_digits</code> . If blank, 0's are used.

Target_lock_digit key/value pairs

team	Unique name shared with <code>target_lock</code>
count	Position of this <code>target_lock_digit</code> within the combination number. 1 = 1's place, 2 = 10's place, etc.

Spawnflags

CROSSLEVEL (=1)

If you set the crosslevel spawnflag for **both** target_lock and target_lock_code, these entities may be placed in different maps **within the same unit**. If this spawnflag is set, the target_lock initially generates a random combination and stores a copy of that combination in global game information that is available to other maps (again, **within the same unit only**). The target_lock_code with the crosslevel spawnflag then reports that combination. The target_lock will not change its combination unless a) the player starts a map with the map command, or leaves the current unit (by triggering a target_changelevel whose map field begins with *). To use a target_lock with the crosslevel feature, the map with the target_lock in it **must** be played before the map with the target_lock_code. Otherwise, when the player triggers the target_lock_code he'll receive the message "Lock has not been properly initialized." You may have as many target_locks in a map as you want, but any target_locks in the same unit that use the crosslevel spawnflag will **all** have the same combination.

HUD (=2)

(Removed from Tremor for Lazarus. May be restored if there's interest) If the HUD spawnflag is set, the known digits of the target_lock combination will be shown in the HUD. Using the HUD, you could use one or more target_lock_clues to reveal the combination **without** using the accompanying target_characters.

Lazarus Main Page

Target_lock

Target_lock_digit

Target_lock_code

Target_lock_clue

The combination lock uses several different entities dependent on your objective: All combination locks use one `target_lock` + between one and eight `target_lock_digit`'s. Each `target_lock_digit` displays one digit from the lock combination. The relationship between `target_lock` and `target_lock_digit` is very similar to `target_string` and `target_character`. The `target_lock` and each `target_lock_digit` must have the same "team" value. `Target_lock` should have a `targetname` and a `target`. In practice, a `trigger_multiple` or `func_button` will generally be used to target the `target_lock`. If the correct combination has been entered with the `target_lock_digits`, `target_lock` fires its `target`. If the correct combination has **not** been entered, `target_lock` triggers its `pathtarget` (if used) or plays the annoying `talk1.wav` we all love and displays its message, if one is specified. The initial display of the `target_lock_digits` is specified with the `key_message` field of `target_lock`. If unspecified, the digits will initially display all 0's. At startup, the game generates a random number and uses that number as the combination. This gives the `target_lock` a bit of replay value.

`Target_lock_digit` is a brush model that should use the `test/num_0` texture, or similar animated texture if you don't care for this one. Along with the "team" value, each `target_lock_digit` should have a unique "count" value. The count value specifies the position of the digit within the combination string. 1 is the left-most digit. You may use as many `target_lock_digits` as you wish, but more than 6 or so is rather pointless. The display is incremented by looking at the `target_lock_digit` and pressing `+use`.

The combination to a `target_lock` may be revealed to the player in several ways. The simplest method is the use of a `target_lock_code`. When triggered, the `target_lock_code` reveals the combination of the associated `target_lock`. The `target` of `target_lock_code` should be the `targetname` of the `target_lock`.

Alternatively, you can use one or more `target_lock_clue`'s. A `target_lock_clue` reveals one or more digits from the `target_lock` combination. The digits that will **not** be revealed are represented by a ? in the `target_lock_clue` message string. For example, if the message string is "X?X?", then the 1st and 3rd digits will be revealed. If the `target_lock_clue` is teamed with `target_characters`, those `target_characters` will be used to display the revealed digits of the lock combination. Initially each of the `target_characters` will scroll continuously.

Target_lock key/value pairs

targetname	Name of the lock.
team	Unique name shared by associated <code>target_lock_digits</code>
target	Name of the entity to trigger if the correct combination has been entered
pathtarget	Name of the entity to trigger if the correct combination has not been entered. If blank, the <code>target_lock</code> plays <code>talk1.wav</code>
message	Text to display if the <code>target_lock</code> is triggered with the incorrect combination.
key_message	Initial combination displayed by <code>target_lock_digits</code> . If blank, 0's are used.

Target_lock_digit key/value pairs

team	Unique name shared with <code>target_lock</code>
count	Position of this <code>target_lock_digit</code> within the combination number. 1 = 1's place, 2 = 10's place, etc.

Spawnflags

CROSSLEVEL (=1)

If you set the crosslevel spawnflag for **both** target_lock and target_lock_code, these entities may be placed in different maps **within the same unit**. If this spawnflag is set, the target_lock initially generates a random combination and stores a copy of that combination in global game information that is available to other maps (again, **within the same unit only**). The target_lock_code with the crosslevel spawnflag then reports that combination. The target_lock will not change its combination unless a) the player starts a map with the map command, or leaves the current unit (by triggering a target_changelevel whose map field begins with *). To use a target_lock with the crosslevel feature, the map with the target_lock in it **must** be played before the map with the target_lock_code. Otherwise, when the player triggers the target_lock_code he'll receive the message "Lock has not been properly initialized." You may have as many target_locks in a map as you want, but any target_locks in the same unit that use the crosslevel spawnflag will **all** have the same combination.

HUD (=2)

(Removed from Tremor for Lazarus. May be restored if there's interest) If the HUD spawnflag is set, the known digits of the target_lock combination will be shown in the HUD. Using the HUD, you could use one or more target_lock_clues to reveal the combination **without** using the accompanying target_characters.

Lazarus Main Page

Target_lock

Target_lock_digit

Target_lock_code

Target_lock_clue

The combination lock uses several different entities dependent on your objective: All combination locks use one `target_lock` + between one and eight `target_lock_digit`'s. Each `target_lock_digit` displays one digit from the lock combination. The relationship between `target_lock` and `target_lock_digit` is very similar to `target_string` and `target_character`. The `target_lock` and each `target_lock_digit` must have the same "team" value. `Target_lock` should have a `targetname` and a `target`. In practice, a `trigger_multiple` or `func_button` will generally be used to target the `target_lock`. If the correct combination has been entered with the `target_lock_digits`, `target_lock` fires its `target`. If the correct combination has **not** been entered, `target_lock` triggers its `pathtarget` (if used) or plays the annoying `talk1.wav` we all love and displays its message, if one is specified. The initial display of the `target_lock_digits` is specified with the `key_message` field of `target_lock`. If unspecified, the digits will initially display all 0's. At startup, the game generates a random number and uses that number as the combination. This gives the `target_lock` a bit of replay value.

`Target_lock_digit` is a brush model that should use the `test/num_0` texture, or similar animated texture if you don't care for this one. Along with the "team" value, each `target_lock_digit` should have a unique "count" value. The count value specifies the position of the digit within the combination string. 1 is the left-most digit. You may use as many `target_lock_digits` as you wish, but more than 6 or so is rather pointless. The display is incremented by looking at the `target_lock_digit` and pressing `+use`.

The combination to a `target_lock` may be revealed to the player in several ways. The simplest method is the use of a `target_lock_code`. When triggered, the `target_lock_code` reveals the combination of the associated `target_lock`. The `target` of `target_lock_code` should be the `targetname` of the `target_lock`.

Alternatively, you can use one or more `target_lock_clue`'s. A `target_lock_clue` reveals one or more digits from the `target_lock` combination. The digits that will **not** be revealed are represented by a ? in the `target_lock_clue` message string. For example, if the message string is "X?X?", then the 1st and 3rd digits will be revealed. If the `target_lock_clue` is teamed with `target_characters`, those `target_characters` will be used to display the revealed digits of the lock combination. Initially each of the `target_characters` will scroll continuously.

Target_lock key/value pairs

targetname	Name of the lock.
team	Unique name shared by associated <code>target_lock_digits</code>
target	Name of the entity to trigger if the correct combination has been entered
pathtarget	Name of the entity to trigger if the correct combination has not been entered. If blank, the <code>target_lock</code> plays <code>talk1.wav</code>
message	Text to display if the <code>target_lock</code> is triggered with the incorrect combination.
key_message	Initial combination displayed by <code>target_lock_digits</code> . If blank, 0's are used.

Target_lock_digit key/value pairs

team	Unique name shared with <code>target_lock</code>
count	Position of this <code>target_lock_digit</code> within the combination number. 1 = 1's place, 2 = 10's place, etc.

Spawnflags

CROSSLEVEL (=1)

If you set the crosslevel spawnflag for **both** target_lock and target_lock_code, these entities may be placed in different maps **within the same unit**. If this spawnflag is set, the target_lock initially generates a random combination and stores a copy of that combination in global game information that is available to other maps (again, **within the same unit only**). The target_lock_code with the crosslevel spawnflag then reports that combination. The target_lock will not change its combination unless a) the player starts a map with the map command, or leaves the current unit (by triggering a target_changelevel whose map field begins with *). To use a target_lock with the crosslevel feature, the map with the target_lock in it **must** be played before the map with the target_lock_code. Otherwise, when the player triggers the target_lock_code he'll receive the message "Lock has not been properly initialized." You may have as many target_locks in a map as you want, but any target_locks in the same unit that use the crosslevel spawnflag will **all** have the same combination.

HUD (=2)

(Removed from Tremor for Lazarus. May be restored if there's interest) If the HUD spawnflag is set, the known digits of the target_lock combination will be shown in the HUD. Using the HUD, you could use one or more target_lock_clues to reveal the combination **without** using the accompanying target_characters.

Lazarus Main Page

Target_monitor

The target_monitor is a new **Lazarus** point entity which switches the player's view to that of the target_monitor itself. In appearance, this is similar to the function of an info_player_intermission, except that the target_monitor may be triggered at any time, and its view is used by only the player that triggered it.

This entity can be used any time you want to show the player something. For example, if the player presses a button that opens a door elsewhere in the map, rather than displaying a cliched "A door has opened somewhere" message, you can let the player see and hear the door actually opening. Another possible use is as a sort of cut-scene camera. The target_monitor viewpoint can be made to move about if it is set to **movewith** a moving parent entity, and the viewpoint can be set to watch another entity by **targeting** it. Alternatively, the target_monitor can act as a chase cam which follows the target entity by a set distance (**CHASE_CAM** spawnflag) or take on the POV of the targeted entity (**EYEBALL** spawnflag).

When tracking a **targeted** entity, the view through the target_monitor may appear jerky as the view angle is updated. If it's a smooth panning action you're after, you might be better served by giving the monitor an **angles** value instead, and setting it to **movewith** a rotating train with its "turn_rider" option enabled.

In much the same way as with the **func_monitor**, when the target_monitor is in use, a "fake player" is spawned at the location the player was when the monitor was triggered. But in the case of the target_monitor, the fake player is not solid (can't be hurt) while remaining not clippable. If a monster is mad at the player when a target_monitor is triggered, the monster will stay mad at the fake player (this keeps the monster from running off trying to figure out where the player went). Since the fake player is not solid, the monster won't be trying to kill it, and possibly damage/kill himself while the player's viewpoint is elsewhere. This ensures that the show you want to put on for the player by using a target_monitor won't be interrupted.

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0. Ignored if a **target** value exists.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0. Ignored if a **target** value exists.

count

When non-zero, specifies the number of times the target_monitor will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

distance

Sets the distance of the viewpoint from the **targeted** entity. Ignored if **CHASE_CAM** is not set. Default=128.

height

Sets the vertical distance of the viewpoint above or below the top of the bounding box of the **targeted** entity. Ignored if **CHASE_CAM** is not set. Default=16.

movewith

Targetname of the parent entity the monitor is to **movewith**.

noise

Specifies the path and filename of the .wav file to play when the view shifts. This sound is only heard by the player that triggers the monitor.

target

Targetname of the entity the monitor will aim at.

targetname

Name of the specific monitor.

wait

Time in seconds for the monitor to wait before it returns the player view to normal. If wait=-1, the monitor must be retriggered for this to happen. Default=3.

Spawnflags**CHASE_CAM Spawnflag (=1)**

The target_monitor will act as a chase cam and will follow its **targeted** entity as specified by **distance** and **height**. The physical location of CHASE_CAM target_monitors in the map is irrelevant.

EYEBALL Spawnflag (=2)

Very similar to the CHASE_CAM spawnflag, except that the viewpoint will be that of the **target** entity (distance=0, height=viewheight of the target). This spawnflag overrides CHASE_CAM. As with CHASE_CAM, the physical location of the target_monitor is irrelevant. The target entity will be made temporarily invisible while the target_monitor is in use so that the player's view doesn't take on the normal orange hue associated with being embedded in another entity. For this reason, EYEBALL isn't appropriate for use in multiplayer games, since the target entity will suddenly become visible/invisible to other players.

NOT_IN_EASY Spawnflag (=256)

The target_monitor will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_monitor will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_monitor will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_monitor will be inhibited and not appear when deathmatch=1.

Target_monsterbattle

This entity is very similar to a Rogue mission pack-style [target_anger](#), but when used in conjunction with the [dmgteam](#) monster key can be used to initiate a full-scale battle between rival monster gangs with a minimum of fuss. If both the **target** and **killtarget** entities are monsters, and these monsters use different **dmgteam** keys, the sequence will go something like this: **target** monster attacks **killtarget** monster, which angers all monsters that have the same **dmgteam** key as the **killtarget**, so they attack the **target** monster. This action in turn angers all monsters with the same **dmgteam** key as the **target** monster, and... well... you get the picture.

Key/Value pairs

target

Targetname of the entity or entities to make angry.

killtarget

Name of the entity that the monster(s) will become angry at.

count

When non-zero, specifies the number of times the target_monsterbattle will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

[Lazarus Main Page](#)

Target_movewith

The **Lazarus** target_movewith point entity is a mechanism which allows **movewith** children to be assigned to or detached from a parent while the game is running. As a couple of examples, this can be used for simulating an automatic crane picking up or dropping something, or freeing monsters from a moving brush model that they're riding.

By default the target_movewith attempts to **attach** a **targeted** child to a **pathtargeted** parent. Setting the **DETACH** spawnflag will cause the target_movewith to **separate** the child from its parent instead.

In practice, there's a propensity for there to be a delay between the CPU doing calculations and the actual gamestate that results. Therefore it's pretty obvious that **attaching** a child to a parent is best done when both child and parent are inactive and not moving so as to avoid this. Naturally, knowing where both entities are at the time the attachment is made would be a requirement to avoid odd-appearing behavior. Conversely, **detaching** a child from its parent can be done pretty much at any time.

Key/value pairs

count

When non-zero, specifies the number of times the target_movewith will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

pathtarget

Targetname of the **movewith** parent entity the child is to be attached to. Ignored if **DETACH** is set.

target

Targetname of the **movewith** child entity to be attached to or detached from a parent entity.

targetname

Name of the specific target_movewith.

Spawnflags

DETACH Spawnflag (=1)

Specifies the target_movewith will detach a child from its parent rather than attach it.

NOT_IN_EASY Spawnflag (=256)

The target_movewith will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_movewith will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_movewith will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_movewith will be inhibited and not appear when deathmatch=1.



Target_playback

The **Lazarus** target_playback is a point entity which is similar to the regular Quake2 target_speaker, except that it supports the playback of sound files in formats other than .WAV, including .MP3, .MID, .MOD, .S3M, and .XM. Its use requires the inclusion of **FMOD.DLL**, which is included in the **Lazarus** files on the [downloads](#) page. For more info about **FMOD**, go [here](#).

For version control reasons (both our version control, and the version control of the hapless user who downloads and runs add-on maps), FMOD.DLL must be placed in your game directory, *not* the root Quake2 folder, and *certainly* not in a system folder. The sound file to be played **cannot** be played from a pakfile location; therefore it must be a loose file. Target_playback looks in the root game directory for files, rather than in the /sound/ directory as a target_speaker does.

For .MP3, .WAV, and .MID files you can have positioned sounds using the **3D** spawnflag. If the 3D spawnflag is **not** set, or if the sound is a music file (.MOD, .S3M, .XM) then the sound just plays back, from everywhere, at the same volume. FMOD.DLL is not loaded unless either a map which includes a target_playback is loaded, or when a sound is manually played (see next paragraph).

To help you test the viability of your custom sound files, **Lazarus** offers the use of the [playsound](#) console command.

FMOD does not get along at all with Quake2's use of the DirectSound primary buffer (s_primary=1). If s_primary is set to 1, Lazarus will display a nag message whenever FMOD functions are called, reminding the player to switch settings.

For loading a game that was saved while a target_playbacks is active, the code will restart that target_playback after the game loads. There is no way to determine the portion of a sound file that has already been played back, so there's no way to "pick up" where you left off. While not a perfect solution, it is better than simply ignoring active target_playbacks, and it preserves any multiple playback sequences set up by using their [targets](#).

Another caveat concerns multiplayer games. The Lazarus gamex86.dll only runs on the server machine, not on the client. As a result of that, there's no way for a server to force a client to use FMOD.DLL, so for multiplayer games, this feature is disabled to avoid unnecessarily burdening the server. It can be turned on with the [packet_fmod_playback](#) server-side cvar, so you can still listen to .MP3's while railing bots.

Reiterating the above, as outlined in the [redistribution](#) doc, this entity will require the following file:

- [fmod.dll](#)

FMOD.DLL is distributed with Lazarus with permission from [Firelight Multimedia](#).

Key/value pairs

count

When non-zero, specifies the number of times the target_playback will be called before being auto-killtargeted (see [this page](#) for details). Default=0. Ignored if **LOOP** is set.

distance

Distance at which 3D sounds begin to attenuate. This value is ignored if **3D** is not set, and is also ignored for .MP3 files. Default value = 40.

fadein

Time in seconds to fade a sound from silence to full **volume**. Default = 0 (no fade).

fadeout

Time in seconds to fade a **LOOP** or **TOGGLE** target_playback from full **volume** to silence when it is triggered off. Default = 0 (no fade).

movewith

Targetname of the parent entity the target_playback is to [movewith](#).

noise

Specifies the path and filename of the sound file to play. The filename should include the extension as well. Target_playback will look in the root game directory for sound files, rather than in /sound/.

target

Targetname of the entity to be triggered upon the completion of the playback in progress. Useful for making one target_playback trigger another, so a series of music tracks may be played in sequence.

targetname

Name of the specific target_playback.

volume

Specifies the volume level of the sound played. Values are 0.0-1.0. Default=1.

Spawnflags**LOOP Spawnflag (=1)**

The target_playback will loop its sound when triggered. Trigger it again to turn it off.

TOGGLE Spawnflag (=2)

If set, if the target_playback is currently playing a sound then it will stop playing when triggered. If TOGGLE is not set, then under the same circumstances the sound would stop and restart. This spawnflag is ignored if LOOP is set since that is the default behavior for LOOP.

MUSIC Spawnflag (=4)

This spawnflag provides the mapper with a means of distinguishing between background music and more critical voiceovers or other sounds. If MUSIC is set and the **fmod_nomusic** cvar is set, this target_playback will not play.

START_ON Spawnflag (=8)

If set, playback begins when the level loads.

3D Spawnflag (=16)

If set, the target_playback's position and velocity and the player's position and velocity (SP mode only) are taken into account. This spawnflag is inapplicable for music (.MOD, .S3M, .XM) files. To control the rate of attenuation, use the global worldspawn **attenuation** value. You can control the Doppler pitch variation with the worldspawn **shift** value. Note that the Doppler setting is universal and will be used for all 3D sounds. You'll probably note that music sounds... very strange if Doppler shift is applied. If **SAMPLE** is also set, then the entire sound file is loaded into memory (SAMPLE is set by default for all but .MP3 files). This allows you to use another FMOD feature to control the distance at which the sound first starts to attenuate (see **distance** value).

Another consideration when using 3D sounds that may not be obvious involves the use of the **movewith** key. If the worldspawn **shift** value is non-zero (meaning Doppler shift effects will be used), then you'll most likely want to place the target_playback very close to the origin of its movewith parent, if the parent rotates rapidly. Otherwise inappropriate Doppler effects will most likely result when the movewith parent rotates.

SAMPLE Spawnflag (=128)

Forces FMOD to load a **3D** sound as a sample rather than as a stream. This will cause the code to precache the sound at startup and allows the proper use of the **distance** value to help control sound attenuation. This spawnflag is set automatically for 3D sounds other than .MP3 files. We do **not** recommend using this spawnflag on large .MP3 files. If SAMPLE is **not** set for an .MP3 file, then the distance value is ignored and a default of about 40 game units is used.

NOT_IN_EASY Spawnflag (=256)

The target_playback will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_playback will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_playback will be inhibited and not appear when skill=2 or greater.

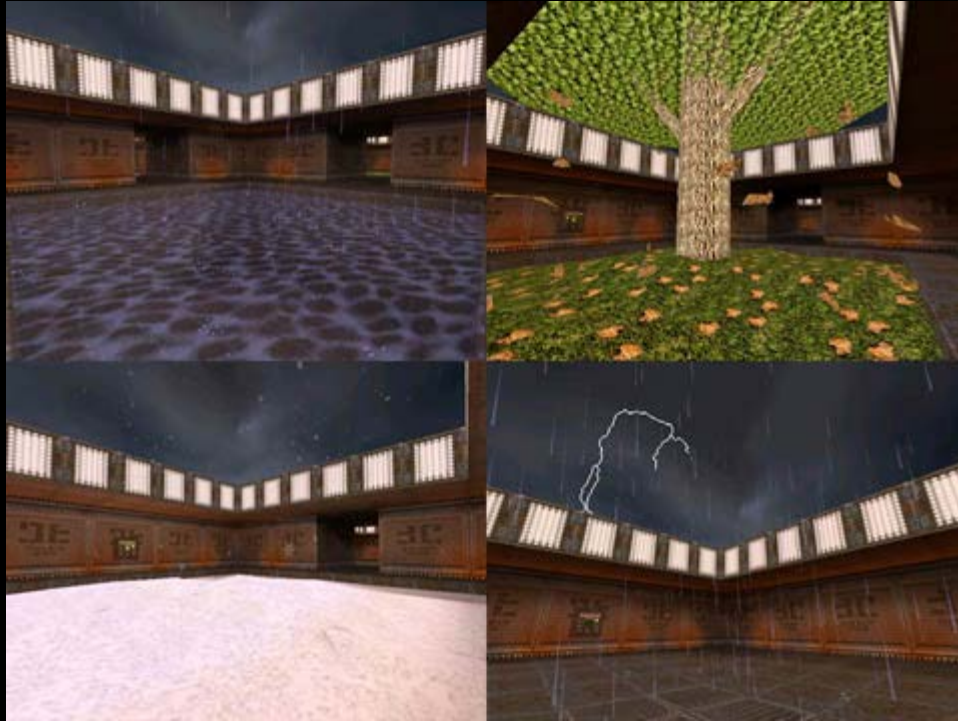
NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_playback will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Target_precipitation

The **Lazarus** target_precipitation allows you to add falling rain, snow, leaves, or a user-defined model to a map for a single player game (it is automatically removed in deathmatch and coop games). Thanks to **Rroff** (Martin Painter) for providing the basis to this code.



How it works: Every 0.1 second frame, target_precipitation spawns a specified number of models with origins located at random positions somewhere within an offset from the target_precipitation origin. The type of model spawned is controlled with the **style** value; you may choose from a single raindrop, 10 raindrops, 10 snowflakes, a leaf, or a custom model. The number of models generated per second is specified with the **count** and **random** values. The number of models spawned in any frame will be $0.1 * (\text{count} \pm \text{random})$, with the remainder after truncating to an integer carried over to the next frame. The offset of the models from the target_precipitation is controlled by the **bleft** and **tright** values. The models will fall in a direction specified with the **angles** value with a constant **speed** and optional **yaw_speed** (rotational velocity about the Z axis).

The user-defined **style** offers a smorgasbord of variations not available with the standard styles. Want blue-shelled gibs leaving a trail of blaster particles and bouncing when they hit the ground? Well, you can have them, you sicko. See the key/value descriptions below for more information on other parameters available for user-defined precipitation.

WARNINGS:

- It is exceptionally easy to bring the game to a grinding halt with this entity if it is abused. Target_precipitation can generate **A LOT** of entities and can very easily produce **SZ_GetSpace: Overflow** errors. The number of entities that will exist at any given time is a function of **count**, **random**, **speed**, and the distance to fall. Every second, target_precipitation generates $\text{count} \pm \text{random}$ entities. So, for example, if **count**=120 and **random**=40, on average target_precipitation will generate 12 entities every 0.1 second frame. The lifespan of each entity is the time required to fall from the initial position to a solid or liquid brush or a solid entity. If **speed**=300 and target_precipitation is 300 units above the floor, newly spawned entities will live approximately 1 second (assuming **bleft** and **tright** are centered vertically around the target_precipitation). So for the above example there will be between 80 and 160 entities active at any given time. Overflow errors have nothing to do with the complexity of a model; the problem is caused by passing too much information from server to client (and yeah, that applies to single player games as well as deathmatch). So think of it this way... instead of 120 raindrops imagine 120 monsters falling from the sky. You should always thoroughly test target_precipitation under adverse conditions before deciding that it's good to go. Give yourself a hyperblaster and invite any available monsters to dance in the area where the target_precipitation is

located. If the map survives that test without generating errors, you're probably safe.

- Along with potential overflow problems, you'll also need to concern yourself with the number of entity polygons (epoly) that are visible. This is more of a concern with the 10-raindrop and 10-snowflake models than the others, since if you generate too many polygons with the single raindrop or leaf models, well you've probably already frozen the game with overflow errors. Particularly pay attention to the epoly count with the 10-snowflake model. Although the model itself is very simple (20 triangles visible for 10 snowflakes), since you'll typically use a slow **speed** value for snow you'll have quite a few snow models present at any given time. Of course the higher the models are dropped from, the more entities you'll have active for a given speed. There is no general consensus on an upper limit for epoly (at least none that we're aware of), but as with all things related to Q2 you'll need to thoroughly test your map on the lowest-end system you can find. If you limit epoly to less than 4000 you'll likely have no problems.
- The snow model uses a color palette trick to make parts of the model completely transparent while using a minimum number of triangles (2 per face). Unfortunately this trick does not work at all using software rendering. Software users won't have any serious problems, but the clear portions of the snowflakes will either be pink or solid white.

The code performs a few tricks to help prevent SZ_GetSpace:Overflow errors that you should be aware of when designing your map:

- Target_precipitation will **not** start operating before the player spawns into the game, regardless of whether **START_ON** is set or not.
- Likewise, target_precipitation will wait until the player spawns into the game after reloading a saved game.
- Target_precipitation will not operate when the player is not in its PVS. If this was **not** done, then a target_precipitation which previously worked fine while the player was in the same area could very easily cause SZ_GetSpace:Overflow errors when the player returned to that area, simply because of the onslaught of entities suddenly becoming visible. While this change is necessary for gameplay, it forces you to pay attention to what you're doing when placing a target_precipitation in the map, so... umm... pay attention :-). You don't want to place a target_precipitation such that the entity would not be in the PVS of the player but the generated effect would be. If you do, then you could very easily create a situation where the player might see rain at one position, but sidestepping a few units causes the rain to stop.

Although the single raindrop model (**style**=0) generally has a better appearance and is easier to implement than the 10-raindrop model (**style**=1), you cannot produce an intense rainstorm using **style**=0 in even a small area. If you want anything more than a drizzle then you should definitely try the 10-raindrop model. However, there are a few considerations to take into account when using this model:

- Individual raindrops that aren't on the model's origin (that's 9 of the 10 drops) will pass through solid objects.
- When the origin hits a surface, all of the model's raindrops will disappear at once.
- Only one impact splash occurs at the origin and not for every individual drop.

The first two problems are also applicable to snow (**style**=2), though not as noticeable. **If** the rain falls on a flat surface with a near-vertical (90 degree) pitch **angle** and there are very few, if any, obstructions between the floor and the sky, then the 10-raindrop model will work fine with no noticeable visual anomalies.

As outlined in the **redistribution** doc, this entity will require these files:

Style	Files
0	models/objects/drop/tris.md2 models/objects/drop/skin.pcx
1	models/objects/drop/heavy.md2 models/objects/drop/skin.pcx
2	models/objects/snow/tris.md2 models/objects/snow/skin.pcx
3	models/objects/leaf1/tris.md2 models/objects/leaf1/skin.pcx models/objects/leaf2/tris.md2 models/objects/leaf2/skin.pcx models/objects/leaf3/tris.md2 models/objects/leaf3/skin.pcx

Key/value pairs

angles

Specifies the direction that the entities will fall. Values are pitch, yaw, and roll (roll is ignored). Positive pitch angles are downward. If omitted, defaults to (90 0 0) (straight down).

attenuation

Controls the rebound of user-defined models after striking a solid. This value is ignored if **style** is not equal to 4, or if **gravity** is 0, or finally if the **BOUNCE** spawnflag is not set. 0=no bounce, 0.9 or so is a SuperBall. 1.0 and higher are physically impossible but still allowed. As a reference point, metallic gibs and grenades use a value of 0.5. Default=0.

bleft

Specifies bottom-left (minX/minY/minZ) bounding box coordinates of the area within which the models will be spawned. See also **tright**. The code will select a random position within this bounding box. Bear in mind that **styles** 1 and 2 cover a 128x128 area. So for those styles you will have raindrops/snowflakes falling approximately 64 units outside the box specified by bleft and tright. Default=(-512 -512 -speed*0.05).

count

Specifies the base number of models spawned each second. The number of entities spawned in any 0.1 second frame will be in the range $0.1 * (\text{count} \pm \text{random})$. Default=1.

effects

Ignored if **style** is not equal to 4. Effects may be combined by adding the values.

- 1: ROTATE Rotate like a weapon
- 2: GIB
- 8: BLASTER - Yellowish orange glow plus particles
- 16: ROCKET - Rocket trail
- 32: GRENADE - Grenade trail
- 64: HYPERBLASTER - BLASTER w/o the particles
- 128: BFG - Big green ball
- 256: COLOR_SHELL
- 512: POWERSCREEN - Green power shield
- 16384: FLIES - Ewwwww
- 32768: QUAD - Blue shell
- 65536: PENT - Red shell
- 131072: TELEPORTER - Teleporter particles
- 262144: FLAG1 - Red glow
- 524288: FLAG2 - Blue glow
- 1048576: IONRIPPER
- 2097152: GREENGIB
- 4194304: BLUE_HB - Blue hyperblaster glow
- 8388608: SPINNING_LIGHTS - Red spinning lights
- 16777216: PLASMA
- 33554432: TRAP
- 67108864: TRACKER
- 134217728: DOUBLE - Yellow shell
- 268435456: SPHERETRANS - Transparent
- 536870912: TAGTRAIL
- 1073741824: HALF_DAMAGE
- 2147483648: TRACKER_TRAIL

fadeout

This value is only applicable to leaves (**style**=3), and user-defined models (**style**=4). Fadeout specifies the time, in seconds, that a model will persist once it touches a solid object before fading away. Keep in mind that the larger fadeout is, the more entities will exist at any given time. Default=0.

gravity

Specifies the ratio of normal gravity that the model will be attracted to the ground with. This value is ignored if **style** is not equal to 4. 1=normal gravity, 0=move at a constant **speed** value. If gravity is non-zero **and** **attenuation** is non-zero **and** the **BOUNCE** spawnflag is set, the model will bounce when it strikes a solid brush, monster, or player. Default=0.

mass

Weight of the user-defined model, used in determining damage to monsters or players struck by the model. Damage inflicted is proportional to the mass times the impact velocity. This value is ignored if **style** is not equal to 4, or if **gravity** is 0. Default = 0.

mass2

Number of splash particles produced when the model strikes a solid object or liquid brush. This value has no effect if **SPLASH** is not set or if **style** is not 0, 1, or 4. Default=8.

pitch_speed

Specifies the maximum angular velocity around the Y axis of the model, in degrees/second. When spawned, the model will be given a constant angular velocity in the range \pm pitch_speed. This value is ignored if **style** is not equal to 4. Default=0.

random

Modifier to **count**. The number of entities spawned per second will be count \pm random. Default=0.

renderfx

Ignored if **style** is not equal to 4. Effects may be combined by adding the values.

- 1: MINLIGHT Never completely dark
- 2: VIEWERMODEL
- 4: WEAPONMODEL
- 8: FULLBRIGHT
- 16: DEPTH HACK
- 32: TRANSLUCENT Transparent
- 64: FRAMELERP
- 128: BEAM
- 512: GLOW Pulsating glow of normal Q2 pickup items
- 1024: SHELL_RED
- 2048: SHELL_GREEN
- 4096: SHELL_BLUE
- 32768: IR_VISIBLE
- 65536: SHELL_DOUBLE
- 131072: SHELL_HALF_DAMAGE White shell

roll_speed

Specifies the angular velocity around the X axis of the model, in degrees/second. When spawned, the model will be given a constant angular velocity in the range \pm roll_speed. This value is ignored if **style** is not equal to 4. Default=0.

sounds

Specifies the color index in the Q2 palette of splash particles. Default=183 (blue)

speed

Speed at which the entities will fall. Default=300 for rain (**style**=0 or 1) and user-defined (**style**=4), and 50 for snow (**style**=2) and leaves (**style**=3).

style

Specifies the type of particles to generate:

- 0: Single raindrop
- 1: 10 raindrops with a 128x128 footprint
- 2: 10 snowflakes with a 128x128 footprint
- 3: Leaf (1 of 3)

4: User-defined

targetname

Name of the specific target_precipitation.

tright

Specifies top-right (maxX/maxY/maxZ) bounding box coordinates of the area within which the models will be spawned. See also **bleft**. The code will select a random position within this bounding box. Bear in mind that **styles** 1 and 2 cover a 128x128 area. So for those styles you will have raindrops/snowflakes falling approximately 64 units outside the box specified by **bleft** and **tright**. Default=(512 512 speed*0.05).

usermodel

Specifies the name of the model to spawn. This value is ignored if **style** is not equal to 4. The filename should be relative to <gamedir>/models for .md2 files, or <gamedir>/sprites for .sp2 files.

yaw_speed

Specifies the angular velocity around the Z axis of the rain or snow models. This can be used to simulate a swirling snow effect. Yaw_speed is ignored for **style**=3 (leaf). Leaves are given a random rotational velocity around all 3 axes to simulate fluttering. For **style**=4 (user-defined), this is the maximum angular velocity; actual velocity will be in the range \pm yaw_speed. Default=0.

Spawnflags

START_ON Spawnflag (=1)

If set, target_precipitation will start when the first player spawns into the map. Otherwise it must be triggered to work.

SPLASH Spawnflag (=2)

If set and **style** is 0, 1, or 4, a splash effect will be generated when the rain or user-defined model touches a solid or liquid brush or a solid entity. Color of the splash particles is controlled by the **sounds** value. Number of splash particles is set with **mass2**.

BOUNCE Spawnflag (=4)

If set, spawned models will bounce when they touch a solid. Bounce physics is determined by the **gravity** setting. This spawnflag is ignored for all but user-defined (**style**=4) models.

FIRE_ONCE Spawnflag (=8)

If set, target_precipitation will produce models only once when triggered, then shut itself off.

START_FADE Spawnflag (=16)

If set and **fadeout** is non-zero, models will begin to fade away when first spawned, rather than waiting until they strike a solid.

NOT_IN_EASY Spawnflag (=256)

The target_precipitation will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_precipitation will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_precipitation will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_precipitation will be inhibited and not appear when deathmatch=1.

Target_rocks

Target_rocks is used to create a landslide. Two new rock models are used here, one with mass=25, the other with mass=100. As an alternative to the models provided with Lazarus, you may specify custom rock models with the **style** value. Players are damaged and translated by falling rocks. See the [rocks.bsp](#) example map.

Key/Value pairs

angles

Direction rocks are thrown. Angles are pitch, yaw, and roll. Pitch is the elevation; negative angles are up. Yaw is the azimuth around the z axis. 0=east, 90=north, etc. Roll is ignored.

count

When non-zero, specifies the number of times the target_rocks will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

mass

Total mass of the rocks generated. The mass value determines how many of what size rock models are spawned, limited to 16 of each. This works much the same way as func_explosive debris does - the math isn't correct but it's simple: a mass of 1600 or greater will give you 16 mass=100 rocks and 16 mass=25 rocks, for a total of mass=2000. Try not to think about it too much :-). Damage to the player is proportional to the mass of the rock that hits him times the impact velocity.

Note: You'll definitely need to thoroughly test a map using target_rocks to ensure that you don't generate SZ_GetSpace Overflow errors, which is very easy to do. If you **do** get that error, decrease the mass or, if you're using multiple simultaneous target_rocks, decrease the number of these entities.

speed

Initial velocity of the rocks, in units/sec.

style

If non-zero, specifies that alternative rock models will be used. Large rock model is "models/objects/rock<style*2+1>/tris.md2" and the small rock model is "models/objects/rock<style*2+2>/tris.md2". Default = 0. Alternative rock models are **not** provided with Lazarus.

targetname

Name of this entity. Target_rocks must be targeted to work.

[Lazarus Main Page](#)

Target_rotation

Target_rotation allows you to change how trigger_multiple, trigger_once, and... well... just about any entity that triggers other entities behaves. Rather than using a single target, target_rotation specifies a list of targets and picks one from the list. Target_rotation acts as a go-between for the trigger entity and the eventual target... in other words you'll target the target_rotation with your trigger_once, and the target_rotation will pick the actual target(s). How the target selection is made is dependent on the spawnflags. If no entity with the selected targetname exists, target_rotation does nothing (it does **not** select another target).

Key/Value pairs

Target

Comma-separated targets to choose from, e.g. "targ1,targ2,targ3" (Do **not** include the quotation marks).

count

When non-zero, specifies the number of times the target_rotation will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

Spawnflags

NO_LOOP (=1)

If set, target_rotation will cycle through the list of targets only once. If not set, the selection process will continuously loop through the target list.

RANDOM (=2)

If set, target_rotation will make a random selection from the list rather than going through the list in order. RANDOM and NO_LOOP cannot be combined.

[Lazarus Main Page](#)

Target_set_effect

The **Lazarus** target_set_effect point entity allows you to set rendering effects for almost any other point entity. In general, target_set_effect has **no** effect on brush models (this isn't strictly true; some effects may be used on brush models if a true OpenGL card is in use, but limiting a map to be playable **only** on OpenGL systems isn't a real swell idea.)

Effects are specified for the **targeted** entity, appropriately enough, with the **effects** and **renderfx** keys. Different effects and renderfx may be combined by adding their values together. Effects may be turned on and off, with or without affecting any other effects that might be active for that entity at the time, by use of the appropriate **style** value.

Note that the normal actions of some entities will immediately wipe out the changes set by target_set_effect. For example, live monsters will remove the RF_SHELL_* renderfx and COLOR_SHELL effects every frame, since those combinations are normally used for monsters being resurrected by a medic or using power armor. If you want a monster to use a color shell, try QUAD, PENT, DOUBLE, or HALF_DAMAGE **effects**.

Key/value pairs

effects

Sets geewhiz rendering effects. Add values together to combine multiple effects. Choices are:

- 1:** ROTATE Rotate like a weapon
- 2:** GIB
- 8:** BLASTER Yellowish orange glow plus particles
- 16:** ROCKET Rocket trail
- 32:** GRENADE Grenade trail
- 64:** HYPERBLASTER BLASTER w/o the particles
- 128:** BFG Big green ball
- 256:** COLOR_SHELL
- 512:** POWERSCREEN Green power shield
- 16384:** FLIES Ewww
- 32768:** QUAD Blue shell
- 65536:** PENT Red shell
- 131072:** TELEPORTER Teleporter particles
- 262144:** FLAG1 Red glow
- 524288:** FLAG2 Blue glow
- 1048576:** IONRIPPER
- 2097152:** GREENGIB
- 4194304:** BLUE_HB Blue hyperblaster glow
- 8388608:** SPINNING_LIGHTS Red spinning lights
- 16777216:** PLASMA
- 33554432:** TRAP
- 67108864:** TRACKER
- 134217728:** DOUBLE Yellow shell
- 268435456:** SPHERETRANS Transparent
- 536870912:** TAGTRAIL
- 1073741824:** HALF_DAMAGE
- 2147483648:** TRACKER_TRAIL

renderfx

Sets eyecandy rendering effects. Add values together to combine multiple effects. Choices are:

- 1: MINLIGHT Never completely dark
- 2: VIEWERMODEL
- 4: WEAPONMODEL
- 8: FULLBRIGHT
- 16: DEPTH HACK
- 32: TRANSLUCENT Transparent
- 64: FRAMELERP
- 128: BEAM
- 512: GLOW Pulsating glow of normal Q2 pickup items
- 1024: SHELL_RED
- 2048: SHELL_GREEN
- 4096: SHELL_BLUE
- 32768: IR_VISIBLE
- 65536: SHELL_DOUBLE
- 131072: SHELL_HALF_DAMAGE White shell
- 262144: USE_DISGUISE

style

Specifies the triggering behavior of the target_set_effect. Choices are:

- 0: COPY - Turn on the specified effects, while turning off all current effects. (Default).
- 1: NOT - Turn off the specified effects, while leaving all current effects alone.
- 2: XOR - Switch the state of specified effects, while leaving all current effects alone.
- 3: OR - Turn on the specified effects, while leaving all current effects alone.

target

Name of the entity to add effects to.

targetname

Name of the specific target_set_effect.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_set_effect will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_set_effect will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_set_effect will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_set_effect will be inhibited and not appear when deathmatch=1.



Target_skill

The **Lazarus** target_skill allows you to force the skill level to Easy, Normal, Hard, or Nightmare. The ideal usage for this entity is in a hub map in which you give the player a choice of skill levels to play, as in the opening map in Quake. The skill level will not change until the next **target_changelevel** event.

There is a bit of room for abuse of this entity. While some may think it a neat trick to change the skill level without the consent of the player, doing so will only irritate most players. This entity should only be used when the player is given the option of which skill level he would like to try, as in the opening sequence of Quake. 'Nuff said :-)

Key/value pairs

count

If non-zero, target_skill will be removed after *count* usages. For more information see the **count** topic.
Default=0.

style

Skill level, 0-3.

targetname

Name of the target_skill.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_skill will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_skill will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_skill will be inhibited and not appear when skill=2 or greater.

Target_sky

The **Lazarus** target_sky point entity changes the environment map seen by the user. Start outside in the morning, enter the (windowless) Strogg base, and exit into the night. Of course this entity is best used when the player cannot see the sky abruptly change.

Key/value pairs

count

When non-zero, specifies the number of times the target_sky will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

sky

Name of the environment map to switch to, e.g. "unit1_".

targetname

Name of the specific target_sky.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_sky will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_sky will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_sky will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_sky will be inhibited and not appear when deathmatch=1.

Target_spawner

The **Lazarus** target_spawner is identical to the standard Quake2 target_spawner point entity, with the addition of **movewith** support, which allows it to move with its parent entity, and the **count** key, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the direction on the XY plane that the spawner will throw its spawned item.

angles

Specifies the direction in 3 dimensions that the spawner will throw its spawned item.

count

When non-zero, specifies the number of times the target_spawner will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

movewith

Targetname of the parent entity the target_explosion is to **movewith**.

speed

Specifies the speed in units/second that the spawner will throw its spawned item.

target

Specifies the classname of the item to spawn.

targetname

Name of the specific target_spawner.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_spawner will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_spawner will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_spawner will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_spawner will be inhibited and not appear when deathmatch=1.



Target_speaker

The **Lazarus** target_speaker is nearly identical to the standard Quake2 target_speaker point entity, with the addition of **movewith** support, which allows it to move with its parent entity, and the **count** key, which allows it to be auto-killtargeted.

Additionally, the **attenuation=-2** option is added, which effectively allows for triggers to use custom sounds.

Target_speaker is limited to playing .WAV files. If you wish to play .MP3 or .MID files instead, then you might want to check out the **target_playback**.

Legend:
Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

attenuation

Specifies how the played sound will attenuate with distance from the speaker origin. Choices are:

-2: No attenuation. The sound is heard throughout the level, and only by the activator of the speaker.

-1: No attenuation. The sound is heard throughout the level.

1: Normal fighting sounds (default).

2: Idle sound level.

3: Ambient sound level.

Attenuation=3 will be used if **LOOPED_ON** or **LOOPED_OFF** are set.

count

When non-zero, specifies the number of times the target_speaker will be called before being auto-killtargeted (see [this page](#) for details). Default=0. Ignored if **LOOPED_ON** or **LOOPED_OFF** are set.

movewith

Targetname of the parent entity the target_speaker is to **movewith**. The origin of the sound depends on the position of the speaker at the time the sound began playing - in other words its origin will not move as it plays.

noise

Specifies the path and filename of the .wav file to play.

targetname

Name of the specific speaker.

volume

Specifies the volume level of the sound played. Values are 0.0-1.0. Default=1.

Spawnflags

LOOPED_ON Spawnflag (=1)

The speaker will loop its sound and is active when the map loads.

LOOPED_OFF Spawnflag (=2)

The speaker will loop its sound and is inactive when the map loads.

RELIABLE Spawnflag (=4)

The speaker will override all other sounds that may be playing.

NOT_IN_EASY Spawnflag (=256)

The target_speaker will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_speaker will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_speaker will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_speaker will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Target_splash

The **Lazarus** target_splash is identical to the standard Quake2 target_splash point entity, with the addition of **movewith** support, which allows it to move with its parent entity.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the direction on the XY plane that the target_splash will throw its pixels.

angles

Specifies the direction in 3 dimensions that the target_splash will throw its pixels.

count

Specifies the number of pixels in each splash, from 1-255. Default=20.

dmg

Specifies the amount of damage hit points the target_splash will generate at its origin. Default=0.

movewith

Targetname of the parent entity the target_splash is to **movewith**.

sounds

Specifies pixel color and effects of the splash. Choices are:

- 1: Sparks.
- 2: Blue water (default).
- 3: Brown water.
- 4: Slime.
- 5: Lava.
- 6: Blood.

targetname

Name of the specific target_splash.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_splash will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_splash will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_splash will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_splash will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Target_temp_entity

The **Lazarus** target_temp_entity is identical to the standard Quake2 target_temp_entity point entity, with the addition of **movewith** support, which allows it to move with its parent entity, and the **count** key, which allows it to be auto-killtargeted.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

When non-zero, specifies the number of times the target_temp_entity will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

movewith

Targetname of the parent entity the target_temp_entity is to **movewith**.

style

Specifies the effect type used. Choices are:

20: TE_BFG_EXPLOSION

21: TE_BFG_BIGEXPLOSION

22: TE_BOSSTPORT (default)

targetname

Name of the specific target_temp_entity.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The target_temp_entity will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_temp_entity will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_temp_entity will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_temp_entity will be inhibited and not appear when deathmatch=1.

Target_text

The target_text point entity is similar in function to the familiar target_help entity, with a few notable exceptions.

- The text screen is brought up immediately when target_text is triggered, rather than by using the "help" command (normally bound to F1).
- The text screen is **only** displayed once per triggering event. Turning it off requires that the user presses the ESC key.
- Only the text message will be displayed, rather than also displaying the monster/kill/secret/goal counts.
- Text may be optionally retrieved from an external file, by setting the **FILE** spawnflag. Using an external file for the text has the advantage of unlimited length (unlike the typical "message" value, which is limited to 128 characters), and of course may be edited without recompiling the map.

As outlined in the **redistribution** doc, if the **NO_BACKGROUND** spawnflag is **not** set, this entity will require the image file **pics/textdisplay.pcx**.

See the **notes** at the end of this page for more details, including the use of optional formatting codes.

Key/value pairs

message

Specifies the string to be displayed when the target_text is triggered. If the **FILE** spawnflag is set, then this key specifies the filename of the external file to retrieve text from.

targetname

Name of the specific target_text.

Spawnflags

FILE Spawnflag (=1)

The text to be displayed will be retrieved from an external file, rather than use the text string specified by **message**, which would instead specify the filename of that external file. The file should be located in the /MAPS/ directory or in a subdirectory of /MAPS/.

NO_BACKGROUND Spawnflag (=2)

The text will be displayed without a background or plaque.

NOT_IN_EASY Spawnflag (=256)

The target_text will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The target_text will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The target_text will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The target_text will be inhibited and not appear when deathmatch=1.

Formatting Notes

These notes apply to the **target_failure** entity as well as to the target_text.

Text displayed in the target_text window will automatically wrap at the appropriate word space, so it isn't necessary to insert `\n` formatting to insert your own line breaks. The maximum number of characters per line is 35. Strings of more than 35 characters without spaces will be wrapped after the 34th character.

The window displays 22 lines of text at one time (or 23 if there are exactly 23 lines). If the message is longer than this, the window will allow scrolling. For such longer messages, prompts to accomplish this with the `[` and `]` keys will appear at the bottom of the window. (The actual commands are **invprev** and **invnext** - this of course assumes the default key bindings are being used by the user).

Optional text formatting codes. All of the following formatting codes should be located at the beginning of a line of text. Text will automatically wrap at word spaces, and text is left-justified by default.

- `*` Use green characters rather than the normal white.
- `\c` Center this line of text.
- `\n` Line break.
- `\r` Right-justify this line of text.
- `\a` Followed immediately by the name of a .wav file to be played when the text is first displayed. The .wav filename should be the only text on this line, and this formatting code should appear at the very beginning of the message.

Lazarus Main Page

Third-person perspective

Third-person perspective as implemented in **Lazarus** was originally developed by **SKULL**. It can be toggled with the **thirdperson** command, or it can be set with the persistent toggle variable **tpp**.

This is more than just a curiosity that we've added. It serves the function of making **func_pushables** easier to handle. As long as **tpp_auto** is on, **Lazarus** will automatically switch the player's view to a third-person perspective when he starts pushing/pulling a **func_pushable**, and switch back to the normal view when he loses contact with the **func_pushable** (unless thidperson view was already on before he took control of the pushable).

If this is something you **don't** want to happen in your maps, you can set **tpp_auto** to 0 in default.cfg, which you would be distributing along with your maps.

There are admittedly a few problems with third-person perspective: It works reasonably well in open areas, but in tight spots it is.... not good. In tight spots the camera is prone to move into solid brushes... it usually corrects itself sooner or later, but the problem is definitely noticeable. In some cases (when the player is in a particularly tight spot) the view goes nuts - hopping back and forth attempting to find a "good" viewpoint. There's really not a good way around these problems other than designing maps around it (in other words don't build tight spots where **func_pushables** are to be used).

[Lazarus Main Page](#)

Tremor_trigger_multiple

The tremor_trigger_multiple is a brush model which is identical to a standard Quake2 **trigger_multiple**, with the notable difference that it is toggled on/off each time it is targeted. (You lose the ability to fire the trigger's targets indirectly with this change. IOW, if you target a normal, enabled trigger_multiple, it fires its targets. A tremor_trigger_multiple cannot work this way).

This trigger also enjoys an enhancement to the **sounds** key - when sounds=3, the trigger is properly silent with no annoying error message concerning trigger1.wav at map load (this enhancement extends to many other trigger entities under Lazarus as well).

Lazarus **movewith** support allows the trigger to move with a parent entity.

Lastly, the tremor_trigger_multiple is one of a number of **Lazarus** triggers that can detect the presence of a TRACK **turret_breach** when it is under the control of the player. This gives the option to trigger events when the player takes control of such a turret.

Also see the **trigger_bbox** for an equivalent point entity that does not use a brush model.

Key/value pairs

angle

Specifies the facing angle on the XY plane of the trigger. If non-zero, the trigger will only fire if the activator's facing angle is within 90 degrees of the trigger's facing angle. Default=0.

count

When non-zero, specifies the number of times the trigger will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

delay

Specifies the delay in seconds before the trigger will fire after being triggered. Default=0.

killtarget

Targetname of the entity to be removed from the map when the trigger fires.

message

Specifies the character string to print to the screen when the trigger fires.

movewith

Targetname of the parent entity the tremor_trigger_multiple is to **movewith**.

pathtarget

If **target** is a trigger_elevator, specifies the path_corner the targeted func_train is to move to.

sounds

Specifies the sound to be played when the trigger fires. Ignored if there is no **message** value. Choices are:

- 0**: Beep-beep (default).
- 1**: Secret.
- 2**: F1 prompt.
- 3**: Silent.

target

Targetname of the entity to be triggered when the trigger fires.

targetname

Name of the specific tremor_trigger_multiple. Each time the trigger's targetname is called, the trigger is toggled on/off.

wait

Specifies the minimum time in seconds between one firing of the trigger and the next. A value of -1 will limit the trigger to only one firing. Default=0.2.

Spawnflags

MONSTER Spawnflag (=1)

The tremor_trigger_multiple will fire if a monster is present within its field.

NOT_PLAYER Spawnflag (=2)

The tremor_trigger_multiple will not fire if a player is present within its field.

TRIGGERED Spawnflag (=4)

The tremor_trigger_multiple is inactive when the map loads and must be targeted to activate it.

OWNED_TURRET Spawnflag (=16)

The tremor_trigger_multiple will fire if a TRACK [turret_breach](#), which is under the player's control, is present within its field. Please see these [notes](#) for proper construction.

NOT_IN_EASY Spawnflag (=256)

The tremor_trigger_multiple will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The tremor_trigger_multiple will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The tremor_trigger_multiple will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The tremor_trigger_multiple will be inhibited and not appear when deathmatch=1.

Trigger_bbox

Trigger_bbox functions exactly like a [tremor_trigger_multiple](#), but is a point entity rather than a brush model. The extents of the trigger field are set with the [bleft](#) and [tright](#) keys. By eliminating the brush model, trigger_bbox helps you reduce the number of unique models in your map, thereby holding at bay the dreaded **ERROR: Index Overflow**.

This trigger also enjoys an enhancement to the [sounds](#) key - when sounds=3, the trigger is properly silent with no annoying error message concerning trigger1.wav at map load (this enhancement extends to many other trigger entities under Lazarus as well). It is also one of a number of **Lazarus** triggers that can detect the presence of a TRACK [turret_breach](#) when it is under the control of the player. This gives the option to trigger events when the player takes control of such a turret.

The trigger can also be made shootable, if desired. When (and only when) [health](#)>0, the trigger is shootable (while remaining non-solid and non-visible). Since it can't be shot from the inside, it's probably advisable to size its bounding box in such a way that it would have a very small thickness, and its placement should be against a visible solid object to avoid the illogical visual effect of shots being stopped in mid-air.

The trigger_bbox is unique in the only trigger field entity which is affected by the [trigger_transition](#). This is something to keep in mind... if you **don't** want a trigger_bbox to make level changes with the player, make sure to not place its origin inside of the trigger_transition's field. Since the trigger_bbox's trigger field can be manually sized and located relative to its origin (and that origin need not be within the bounding box), you could have the active area of the trigger inside the trigger_transition while its origin is outside, or vice-versa.

And, **Lazarus** [movewith](#) support allows the trigger to move with a parent entity.

Key/value pairs

angle

Specifies the facing angle on the XY plane of the trigger. If non-zero, the trigger will only fire if the activator's facing angle is within 90 degrees of the trigger's facing angle. Default=0.

bleft

Specifies bottom-left (minY/minX/minZ) bounding box coordinates for the active trigger field, using the trigger's origin as the reference. See also [tright](#). Default=(-16 -16 -16).

count

When non-zero, specifies the number of times the trigger will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

delay

Specifies the delay in seconds before the trigger will fire after being triggered. Default=0.

health

When non-zero, makes the trigger shootable, and specifies hit points required before activating. A shootable trigger_bbox does not fire when an entity enters its field, but it does remain non-solid and non-visible. Default=0.

killtarget

Targetname of the entity to be removed from the map when the trigger fires.

message

Specifies the character string to print to the screen when the trigger fires.

movewith

Targetname of the parent entity the trigger_bbox is to [movewith](#).

pathtarget

If **target** is a trigger_elevator, specifies the path_corner the targeted func_train is to move to.

sounds

Specifies the sound to be played when the trigger fires. Ignored if there is no **message** value. Choices are:

- 0**: Beep-beep (default).
- 1**: Secret.
- 2**: F1 prompt.
- 3**: Silent.

target

Targetname of the entity to be triggered when the trigger fires.

targetname

Name of the specific trigger_bbox. Each time the trigger's targetname is called, the trigger is toggled on/off.

tright

Specifies top-right (maxY/maxX/maxZ) bounding box coordinates for the active trigger field, using the trigger's origin as the reference. See also **bleft**.
Default=(16 16 16).

wait

Specifies the minimum time in seconds between one firing of the trigger and the next. A value of -1 will limit the trigger to only one firing. Default=0.2.

Spawnflags**MONSTER Spawnflag (=1)**

The trigger_bbox will fire if a monster is present within its field.

NOT_PLAYER Spawnflag (=2)

The trigger_bbox will not fire if a player is present within its field.

TRIGGERED Spawnflag (=4)

The trigger_bbox is inactive when the map loads and must be targeted to activate it.

OWNED_TURRET Spawnflag (=16)

The trigger_bbox will fire if a TRACK **turret_breach**, which is under the player's control, is present within its field. Please see these **notes** for proper construction.

NOT_IN_EASY Spawnflag (=256)

The trigger_bbox will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_bbox will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_bbox will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_bbox will be inhibited and not appear when deathmatch=1.

Trigger_disguise

The **Lazarus** trigger_disguise is a brush model trigger which is taken directly from **Rogue's** mission pack. It effectively makes the player which touches it invisible to monsters until the player fires a weapon, or until another trigger_disguise with the **REMOVE** spawnflag is touched.

Key/value pairs

count

When non-zero, specifies the number of times the trigger_disguise will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

targetname

Name of the specific trigger_disguise.

Spawnflags

TOGGLE Spawnflag (=1)

The trigger_disguise can be toggled on/off.

START_ON Spawnflag (=2)

The trigger_disguise will be active when the map loads.

REMOVE Spawnflag (=4)

The trigger_disguise will remove the "notarget" effect given to the player by another (non-REMOVE) trigger_disguise.

NOT_IN_EASY Spawnflag (=256)

The trigger_disguise will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_disguise will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_disguise will be inhibited and not appear when skill=2 or greater.

Trigger_fog

The **Lazarus** trigger_fog brush model provides one method of introducing fog effects in your map. Unlike **target_fog**, trigger_fog is used locally. When the player's viewpoint is inside the bounding box of the trigger_fog field, he sees the fog specified by that trigger_fog; when his viewpoint leaves that bounding box, he sees the fog (if any) that he saw before entering that field. The fog effect cannot be viewed from the outside of the fog field - that's not how this stuff operates. If you are unfamiliar with how **Lazarus** fog works, then get the whole scoop on the **Fog Effects** page.

When the player is inside a trigger_fog, the fog setting specified by that trigger_fog will override fog effects specified by a **target_fog**.

Fog can be made directional (the player sees denser fog looking in one direction than he would looking in the opposite direction). This is done by giving **fog_density** and **density** different values, and setting view direction with **angle** or **angles**.

When the player's viewpoint enters a trigger_fog, the fog effect can be made to come on abruptly (useful for underwater fog) or can be made to ramp up/down gradually with the use of the **delay** key.

Key/value pairs

angle

Specifies the player facing direction on the XY plane where he will see the fog density value as specified by **fog_density**. Default=0. Only relevant for "directional" fog; ignored if **density**=0.

angles

Specifies the player facing direction in 3 dimensions, defined by pitch and yaw (roll is ignored), where he will see the fog density value as specified by **fog_density**. Default=0 0 0. Only relevant for "directional" fog; ignored if **density**=0.

count

When non-zero, specifies the number of times the trigger_fog will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

delay

Specifies the time in seconds that the trigger_fog will ramp fog density up/down. Density ramps from the player's currently rendered fog level to the fog density specified by the trigger_fog. Default=0.

density

When non-zero, specifies "directional" fog. Non-zero values specify the level of fog density viewed by the player when his view is directly **opposite** the direction specified by **angle/angles**. For best results, use values of <100. Default=0. Ignored when **fog_model**=0.

fog_color

Specifies relative RGB color components of the fog effect; valid range for each component is 0.0-1.0. Default=0.5 0.5 0.5.

fog_density

Specifies the level of fog density as viewed by the player. If **density** is non-zero, the fog is "directional", and fog_density will specify the fog density of the player when his view is aligned with the direction specified by **angle/angles**. For best results, use values of <100. Default=20. Ignored when **fog_model**=0.

fog_far

Specifies the distance in map units from the player's viewpoint where visibility will be completely obscured by fog. See also **fog_near**. Default=1024. Ignored if **fog_model**>0.

fog_model

Specifies the method used to calculate how fog density increases with distance. Default=1. Note: Linear fog cannot be "directional". Choices are:

0: Linear

1: Exponential

2: Fast exponential

fog_near

Specifies the distance in map units from the player's viewpoint where visibility will first begin to be obscured by fog. See also **fog_far**. Default=20. Ignored if **fog_model**>0.

targetname

Name of the specific trigger_fog.

Spawnflags

TOGGLE Spawnflag (=2)

The trigger_fog can be toggled on and off. Fog density levels will ramp up/down as specified by **delay**.

START_OFF Spawnflag (=8)

The trigger_fog will not be active when the map loads.

NOT_IN_EASY Spawnflag (=256)

The trigger_fog will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_fog will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_fog will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_fog will be inhibited and not appear when deathmatch=1.

Trigger_hurt

The **Lazarus** trigger_hurt is identical to the standard Quake2 trigger_hurt brush model, except for cases where **dmg** is set to a negative value, in order to give the player health rather than take it away. In that event, the effect of the player taking damage is disabled (no spraying blood pixels nor any moaning and groaning). In addition, if the **SILENT** spawnflag is not set, a negative-dmg trigger_hurt will play the small health pickup sound (sound/items/s_health.wav).

The new **NOGIB** spawnflag, as its name implies, will cause the trigger_hurt to stop dealing its damage to victims once they are dead. The **ENVIRONMENT** spawnflag will create a trigger_hurt which will not hurt players wearing the environment suit.

Lazarus **movewith** support allows the trigger to move with a parent entity.

Another change to the original code is a check for the presence of a damageable entity at activation. In standard Q2, if a monster is standing inside a trigger_hurt when it is activated, he won't "touch" the trigger until he moves or reacts to the player.

Legend:
Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

dmg

Specifies the number of damage hit points the trigger_hurt will do every 0.1 seconds. Default=5. If set to a negative value, the trigger_hurt will give health, but will not cause pain effects.

movewith

Targetname of the parent entity the trigger_hurt is to **movewith**.

targetname

Name of the specific trigger_hurt.

Spawnflags

START_OFF Spawnflag (=1)

The trigger_hurt is inactive when the map loads and must be targeted to activate it.

TOGGLE Spawnflag (=2)

The trigger_hurt can be toggled on and off.

SILENT Spawnflag (=4)

No sounds will be played when the trigger_hurt is acting upon the player.

NO_PROTECTION Spawnflag (=8)

The trigger_hurt will affect players with GODmode on.

SLOW_HURT Spawnflag (=16)

The trigger_hurt will deal its **damage** more slowly.

NOGIB Spawnflag (=32)

The trigger_hurt will stop dealing damage when its victim dies, and will not gib him.

ENVIRONMENT Spawnflag (=64)

The trigger_hurt will not damage a player with an active environment suit.

NOT_IN_EASY Spawnflag (=256)

The trigger_hurt will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_hurt will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_hurt will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_hurt will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Trigger_inside

The **Lazarus** trigger_inside is similar to the standard Quake2 trigger_multiple brush model, except that it *only* fires when the bounding box of the **pathtargeted** entity is *completely inside* the trigger's bounding box. If you want a trigger to fire only when a certain monster enters it - or you want a trigger to fire when a certain brush model (like a **func_train** or **func_pushable**) enters it - then this trigger is for you.

To reiterate: The activator must be completely inside the trigger before it fires. If the activator merely touches the trigger field, nothing will happen.

This trigger is ideal for puzzles which require the player to move or otherwise manipulate a series of objects into a set position. Like triggering open Egyptian tombs for example.

This trigger enjoys an enhancement to the **sounds** key - when sounds=3, the trigger is properly silent with no annoying error message concerning trigger1.wav at map load (this enhancement extends to many other trigger entities under Lazarus as well).

And, **Lazarus** **movewith** support allows the trigger to move with a parent entity.

Key/value pairs

angle

Specifies the facing angle on the XY plane of the trigger. If non-zero, the trigger will only fire if the activator's facing angle is within 90 degrees of the trigger's facing angle. Default=0.

delay

Specifies the delay in seconds before the trigger will fire after being triggered. Default=0.

killtarget

Targetname of the entity to be removed from the map when the trigger fires.

message

Specifies the character string to print to the screen when the trigger fires.

movewith

Targetname of the parent entity the trigger_inside is to **movewith**.

pathtarget

Targetname of the entity which must be inside the trigger's bounding box before the trigger will fire. Every trigger_inside must have a pathtarget set.

sounds

Specifies the sound to be played when the trigger fires. Ignored if there is no **message** value. Choices are:

0: Beep-beep (default).

1: Secret.

2: F1 prompt.

3: Silent.

target

Targetname of the entity to be triggered when the trigger fires.

targetname

Name of the specific trigger_inside.

wait

Specifies the minimum time in seconds between one firing of the trigger and the next. A value of -1 will limit the trigger to only one firing. Default=0.2.

Spawnflags

TRIGGERED Spawnflag (=4)

The trigger_inside is inactive when the map loads and must be targeted to activate it.

NOT_IN_EASY Spawnflag (=256)

The trigger_inside will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_inside will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_inside will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_inside will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Trigger_key

The **Lazarus** trigger_key is identical in operation to the standard Quake2 trigger_key point entity, but includes a few coding enhancements. Using a **key_message** value, you can give the player whatever clue you want as to what he needs to get the trigger_key to work. Additionally, the **MULTI_USE**, **KEEP_KEY**, and **SILENT** flags allow for some unconventional uses of this entity.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

delay

Specifies the delay in seconds before the trigger_key will fire after being triggered. Default=0.

item

Classname of the entity required to trigger the trigger_key.

key_message

Character string to be printed to the screen when the trigger_key is tried by a player who lacks the required **item**. If key_message has no value, the default "You need the [key name]" message will be printed to the screen instead. Ignored when the **SILENT** spawnflag is set.

killtarget

Targetname of the entity to be removed from the map when the trigger_key fires.

message

Character string to be printed to the screen when the trigger_key fires. Ignored when the **SILENT** spawnflag is set.

target

Targetname of the entity to be triggered when the trigger_key fires.

targetname

Name of the specific trigger_key.

Spawnflags

MULTI_USE Spawnflag (=1)

The trigger_key will be persistent, in that the required **item** is needed every time the trigger_key is used. This is useful for things like vending machines.

KEEP_KEY Spawnflag (=2)

The trigger_key will not remove the required **item** from the player's inventory when the trigger_key is used.

SILENT Spawnflag (=4)

The trigger_key will not play any sounds nor display messages when used. Useful for removing items and weapons from the player on level changes.

NOT_IN_EASY Spawnflag (=256)

The trigger_key will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_key will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_key will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_key will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Trigger_look

The **Lazarus** trigger_look brush model functions like a **tremor_trigger_multiple**, with the addition that the player must be looking at a *separate* bounding box that the trigger_look *points* to before it will fire its targets. The trick now is to define that bounding box, and that can be done in one of two ways.

The first method is to incorporate an origin brush with the trigger_look, in order to define the location of that separate bounding box. Constructing such a trigger is simple. Make a brush with a trigger texture (this will be the field the player must be standing in for the trigger to work). Then make another brush using the origin texture, and place it where you want the player to be looking before the trigger will fire. Select both these brushes and make them into a single brush model - a trigger_look. Of course, the origin brush need not be within the bounds of the trigger field area. You could easily place a small trigger field at one side of a room and the origin of the bounding box it points to at the other. The size of that bounding box is determined by the values of **bleft** and **tright** - by default it's a 32x32x32 cube, but you can make it as large or as small as you like. Note that very small, skinny bounding boxes, viewed at angles, may not be picked up by the trigger_look. If your defined bounding box is a small one, then the best results are achieved by placing it on the same horizontal plane as that of the player's view origin.

The second method does away with the necessity of the origin brush and defining it with **bleft** and **tright**. Instead, the trigger_look uses the bounding box of the entity it **targets**. To enable this sort of trigger_look, you must set its **LOOK_TARGET** spawnflag. So, if the target of a LOOK_TARGET trigger_look is a func_button, then the bounding box of the func_button will be used. Such a button will not be activated unless the player is standing in the trigger_field and is also looking at the button.

Here's a cool thing about **LOOK_TARGET** trigger_look - you could have it service multiple targeted entities. So a bank of func_buttons, all with identical targetnames, could be triggered independently with the same trigger_look. Only the one you're looking at will be activated, rather than all of them.

Lazarus **movewith** support allows the trigger to move with a parent entity. A bank of buttons can be made to movewith a func_train which is targeted by a trigger_elevator, and a single LOOK_TARGET trigger_look could be made to movewith that train as well. You take it from there.

If desired, you may also require the player to be pressing the +use key to activate the trigger. This effectively gives +use capability to every entity the player can trigger, like buttons.

Like a tremor_trigger_multiple, trigger_look will be toggled on/off each time it is targeted.

This trigger also enjoys an enhancement to the **sounds** key - when sounds=3, the trigger is properly silent with no annoying error message concerning trigger1.wav at map load (this enhancement extends to many other trigger entities under Lazarus as well).

Lastly, the trigger_look is one of a number of **Lazarus** triggers that can detect the presence of a TRACK **turret_breach** when it is under the control of the player. This gives the option to trigger events when the player takes control of such a turret, and is looking in the "proper" direction.

Key/value pairs

bleft

Specifies bottom-left (minY/minX/minZ) coordinates for the bounding box the player must be looking at before the trigger will fire, using the center of the associated origin brush as the reference. See also **tright**. Default=(-16 -16 -16). Ignored when **LOOK_TARGET** is set.

count

When non-zero, specifies the number of times the trigger will be turned off before it is auto-killtargeted (see [this page](#) for details). Default=0.

delay

Specifies the delay in seconds before the trigger will fire after being triggered. Default=0.

killtarget

Targetname of the entity to be removed from the map when the trigger fires.

message

Specifies the character string to print to the screen when the trigger fires.

movewith

Targetname of the parent entity the trigger_look is to **movewith**.

pathtarget

If **target** is a trigger_elevator, specifies the path_corner the targeted func_train is to move to.

sounds

Specifies the sound to be played when the trigger fires. Ignored if there is no **message** value. Choices are:

0: Beep-beep (default).

1: Secret.

2: F1 prompt.

3: Silent.

target

Targetname of the entity to be triggered when the trigger fires. If **LOOK_TARGET** is set, the trigger will only act upon the entity being looked at, rather than all entities with that targetname.

targetname

Name of the specific trigger_look. Each time the trigger's targetname is called, the trigger is toggled on/off.

tright

Specifies top-right (maxY/maxX/maxZ) coordinates for the bounding box the player must be looking at before the trigger will fire, using the center of the associated origin brush as the reference. See also **bleft**. Default=(16 16 16). Ignored when **LOOK_TARGET** is set.

wait

Specifies the minimum time in seconds between one firing of the trigger and the next. A value of -1 will limit the trigger to only one firing. Default=0.2.

Spawnflags

TRIGGERED Spawnflag (=4)

The trigger_look is inactive when the map loads and must be targeted to activate it.

USE Spawnflag (=8)

The trigger_look will require the player to have the +use command active as a condition of the trigger's firing. As a practical matter, this spawnflag is incompatible with **OWNED_TURRET**.

OWNED_TURRET Spawnflag (=16)

The trigger_look will fire if a TRACK **turret_breach**, which is under the player's control, is present within its field, and the player is looking in the defined location. As a practical matter, this spawnflag is incompatible with **USE**. Please see these **notes** for proper construction.

LOOK_TARGET Spawnflag (=32)

The trigger_look will use the entity specified by its **target** to define the bounding box the player needs to be looking at before the trigger will fire; and then it will only trigger the specific entity being looked at, rather than triggering all entities with that targetname. If set, no origin brush is required, and **bleft** and **tright** are ignored.

NOT_IN_EASY Spawnflag (=256)

The trigger_look will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_look will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_look will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_look will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Trigger_mass

The **Lazarus** trigger_mass is similar to the standard Quake2 trigger_multiple brush model, except that it *only* fires when touched by sufficient entity mass. *Any* entity (not just players or monsters/actors) can trigger it - this includes entities like **func_pushables**.



The mass value set for the activating entity must be greater than or equal to the **mass** value of the trigger_mass, or the trigger does nothing. This trigger might be useful for requiring the player to drop or otherwise move a object into position to "dislodge" a stuck lever or break a barrier. Or to simulate a collapsing bridge when a heavy **monster** or **actor** (remember, monster/actor mass is adjustable with **Lazarus**) steps on it. (*As an aside, the Q2 player mass=200.*)

This trigger enjoys an enhancement to the **sounds** key - when sounds=3, the trigger is properly silent with no annoying error message concerning trigger1.wav at map load (this enhancement extends to many other trigger entities under Lazarus as well).

And, **Lazarus** **movewith** support allows the trigger to move with a parent entity.

Key/value pairs

angle

Specifies the facing angle on the XY plane of the trigger. If non-zero, the trigger will only fire if the activator's facing angle is within 90 degrees of the trigger's facing angle. Default=0.

delay

Specifies the delay in seconds before the trigger will fire after being triggered. Default=0.

killtarget

Targetname of the entity to be removed from the map when the trigger fires.

mass

Minimum mass required of the activating entity in order for the trigger to fire. Default=100.

message

Specifies the character string to print to the screen when the trigger fires.

movewith

Targetname of the parent entity the trigger_mass is to **movewith**.

pathtarget

If **target** is a trigger_elevator, specifies the path_corner the targeted func_train is to move to.

sounds

Specifies the sound to be played when the trigger fires. Ignored if there is no **message** value. Choices are:

0: Beep-beep (default).

1: Secret.

2: F1 prompt.

3: Silent.

target

Targetname of the entity to be triggered when the trigger fires.

targetname

Name of the specific trigger_mass.

wait

Specifies the minimum time in seconds between one firing of the trigger and the next. A value of -1 will limit the trigger to only one firing. Default=0.2.

Spawnflags**TRIGGERED Spawnflag (=4)**

The trigger_mass is inactive when the map loads and must be targeted to activate it.

NOT_IN_EASY Spawnflag (=256)

The trigger_mass will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_mass will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_mass will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_mass will be inhibited and not appear when deathmatch=1.

Trigger_monsterjump

The **Lazarus** trigger_monsterjump is identical to the standard Quake2 trigger_monsterjump brush model, with the addition of **movewith** support, which allows it to move with its parent entity.

Legend:

Compared to Standard Quake2, Key/Flag is:

New

Modified

Unchanged

Key/value pairs

angle

Specifies the direction angle on the XY plane that the trigger_monsterjump will cause monsters/actors to jump. Default=0.

height

Vertical velocity in units/sec that monsters/actors will be thrown upwards. Default=200.

movewith

Targetname of the parent entity the trigger_monsterjump is to **movewith**.

speed

Horizontal velocity in units/sec that monsters/actors will be thrown, in the direction specified by **angle**. Default=200.

targetname

Name of the specific trigger_monsterjump.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The trigger_monsterjump will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_monsterjump will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_monsterjump will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_monsterjump will be inhibited and not appear when deathmatch=1.

Trigger_multiple

The **Lazarus** trigger_multiple is identical to the standard Quake2 trigger_multiple brush model, with the following changes:

This trigger enjoys an enhancement to the **sounds** key - when sounds=3, the trigger is properly silent with no annoying error message concerning trigger1.wav at map load (this enhancement extends to many other trigger entities under Lazarus as well).

Lazarus **movewith** support allows the trigger to move with a parent entity.

Also, the trigger_multiple is one of a number of **Lazarus** triggers that can detect the presence of a TRACK **turret_breach** when it is under the control of the player. This gives the option to trigger events when the player takes control of such a turret.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the facing angle on the XY plane of the trigger. If non-zero, the trigger will only fire if the activator's facing angle is within 90 degrees of the trigger's facing angle. Default=0.

delay

Specifies the delay in seconds before the trigger will fire after being triggered. Default=0.

killtarget

Targetname of the entity to be removed from the map when the trigger fires.

message

Specifies the character string to print to the screen when the trigger fires.

movewith

Targetname of the parent entity the trigger_multiple is to **movewith**.

pathtarget

If **target** is a trigger_elevator, specifies the path_corner the targeted func_train is to move to.

sounds

Specifies the sound to be played when the trigger fires. Ignored if there is no **message** value. Choices are:

0: Beep-beep (default).

1: Secret.

2: F1 prompt.

3: Silent.

target

Targetname of the entity to be triggered when the trigger fires.

targetname

Name of the specific trigger_multiple. Each time the trigger's targetname is called and the trigger is already active, the trigger will fire.

wait

Specifies the minimum time in seconds between one firing of the trigger and the next. A value of -1 will limit the trigger to only one firing. Default=0.2.

Spawnflags

MONSTER Spawnflag (=1)

The trigger_multiple will fire if a monster is present within its field.

NOT_PLAYER Spawnflag (=2)

The trigger_multiple will not fire if a player is present within its field.

TRIGGERED Spawnflag (=4)

The trigger_multiple is inactive when the map loads and must be targeted to activate it.

OWNED_TURRET Spawnflag (=16)

The trigger_multiple will fire if a TRACK **turret_breach**, which is under the player's control, is present within its field. Please see these **notes** for proper construction.

NOT_IN_EASY Spawnflag (=256)

The trigger_multiple will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_multiple will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_multiple will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_multiple will be inhibited and not appear when deathmatch=1.

Trigger_once

The **Lazarus** trigger_once is identical to the standard Quake2 trigger_once brush model, with the following changes:

This trigger enjoys an enhancement to the **sounds** key - when sounds=3, the trigger is properly silent with no annoying error message concerning trigger1.wav at map load (this enhancement extends to many other trigger entities under Lazarus as well).

Lazarus **movewith** support allows the trigger to move with a parent entity.

Also, the trigger_once is one of a number of **Lazarus** triggers that can detect the presence of a TRACK **turret_breach** when it is under the control of the player. This gives the option to trigger events when the player takes control of such a turret.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the facing angle on the XY plane of the trigger. If non-zero, the trigger will only fire if the activator's facing angle is within 90 degrees of the trigger's facing angle. Default=0.

delay

Specifies the delay in seconds before the trigger will fire after being triggered. Default=0.

killtarget

Targetname of the entity to be removed from the map when the trigger fires.

message

Specifies the character string to print to the screen when the trigger fires.

movewith

Targetname of the parent entity the trigger_once is to **movewith**.

pathtarget

If **target** is a trigger_elevator, specifies the path_corner the targeted func_train is to move to.

sounds

Specifies the sound to be played when the trigger fires. Ignored if there is no **message** value. Choices are:

- 0**: Beep-beep (default).
- 1**: Secret.
- 2**: F1 prompt.
- 3**: Silent.

target

Targetname of the entity to be triggered when the trigger fires.

targetname

Name of the specific trigger_once. If the trigger's targetname is called and the trigger is already active, the trigger will fire.

Spawnflags

TRIGGERED Spawnflag (=4)

The trigger_once is inactive when the map loads and must be targeted to activate it.

OWNED_TURRET Spawnflag (=16)

The trigger_once will fire if a TRACK **turret_breach**, which is under the player's control, is present within its field. Please see these **notes** for proper construction.

NOT_IN_EASY Spawnflag (=256)

The trigger_once will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_once will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_once will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_once will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Trigger_push

The **Lazarus** trigger_push brush model differs from that of the standard game in that it allows you can specify your own sound effect (or none) with the use of the **CUSTOM_SOUND** spawnflag and **noise** key instead of using the default wind sound normally played. Trigger_push will throw players, grenades, and **func_pushables**.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies the direction angle on the XY plane that the trigger_push will throw other entities. Default=0.

angles

Specifies the direction angle in 3 dimensions that the trigger_push will throw other entities, defined by pitch, yaw, and roll. Default=0 0 0.

movewith

Targetname of the parent entity the trigger_push is to **movewith**.

noise

Specifies the path and filename of the .wav file to play when the trigger_push is acting on an entity. If no file is specified, no sound will be played. Ignored if **CUSTOM_SOUND** is not set.

speed

Velocity in [units/sec]*10 that the affected entity will be thrown. Maximum speed allowed by the executable is 4096 units/sec, so the standard game default value of 1000 is obviously quite goofed up. Set to something more logical. Default=1000.

targetname

Name of the specific trigger_push.

Spawnflags

PUSH_ONCE Spawnflag (=1)

The trigger_push will remove itself after its first use.

CUSTOM_SOUND Spawnflag (=2)

The trigger_push will call a custom .wav file when acting on an entity. This file is specified with the **noise** key.

NOT_IN_EASY Spawnflag (=256)

The trigger_push will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_push will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_push will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_push will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Trigger_relay

The **Lazarus** trigger_relay is identical to the standard Quake2 trigger_relay point entity, with the addition of **count** support, which allows it to be auto-killtargeted, and the use of the **dmgteam** key, which creates the ability to use the relay to trigger events when a monster or actor is injured.

Also, this trigger enjoys an enhancement to the **sounds** key - when sounds=3, the relay is properly silent with no annoying error message concerning trigger1.wav at map load (this enhancement extends to many other trigger entities under Lazarus as well).

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

count

When non-zero, specifies the number of times the trigger_relay will be called before being auto-killtargeted (see [this page](#) for details). Default=0.

delay

Specifies the delay in seconds before the relay will fire after being triggered. Default=0.

dmgteam

Specifies the dmgteam value. If a monster_* or actor with an identical dmgteam value is injured, the relay will fire.

killtarget

Targetname of the entity to be removed from the map when the relay fires.

message

Specifies the character string to print to the screen when the relay fires.

pathtarget

If **target** is a trigger_elevator, specifies the path_corner the targeted func_train is to move to.

sounds

Specifies the sound to be played when the relay fires. Ignored if there is no **message** value. Choices are:

0: Beep-beep (default).

1: Secret.

2: F1 prompt.

3: Silent.

target

Targetname of the entity to be triggered when the relay fires.

targetname

Name of the specific trigger_relay.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The trigger_relay will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_relay will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_relay will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_relay will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Trigger_scales

The **Lazarus** trigger_scales is a brush model trigger which doesn't actually trigger anything. What it does is cause changes in a numerical character display as entity mass within its bounds changes. In plain English, it's a weight scale.



To create the numerical display, the trigger_scales must be **teamed** with one or more target_characters. The trigger will report the total mass of all entities within its field, and that number will be represented by the numerals shown by the target_characters. Yes that means multiple entities can be stacked, and their total weight reported. Fractional mass of an entity will be reported as well: if an entity is partially inside and partially outside the trigger_scales, only the portion of the entity's volume inside the trigger will be used to report that portion of the entity's mass.

To properly set the "count" value of the target_characters, realize that the target_characters are positioned from right-to-left. So the target_character with count=1 would give a readout of the 1's place, count=2 gives a readout of the 10's place, and so on.

By itself this isn't a particularly useful entity, but is a nice effect for a warehouse-type environment.

Key/value pairs

targetname

Name of the specific trigger_scales.

team

Identical to the team value of the target_characters used to create the weight display. Target_character "count" values are from right-to-left, in other words, count=1 is the 1's place, count=2 is the 10's place, etc.

Spawnflags

TRIGGERED Spawnflag (=4)

The trigger_scales is inactive when the map loads and must be targeted to activate it.

NOT_IN_EASY Spawnflag (=256)

The trigger_scales will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_scales will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_scales will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_scales will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Trigger_teleporter

The **Lazarus** trigger_teleporter is a brush model entity which may be used as an alternative to the **misc_teleporter**. It is a nodraw trigger with its bounding box extents defined by the size of the brush used, like with any other trigger. Note that all Quake2 triggers use cubical bounding boxes to define trigger fields; don't expect a cylindrical brush to produce a cylindrical trigger field.

You can specify whether or not you want to use the normal teleporter effects with the **NO_EFFECTS** spawnflag, and you can specify your own teleporter sound with the use of the **CUSTOM_SOUND** spawnflag and **noise** key. Note that in order for the trigger_teleporter to properly display its "teleport away" particle effect from the desired location, you should add an origin brush to it. Otherwise, that effect will be displayed at the map origin (0 0 0).

The **LANDMARK** flag allows the teleporting player to inherit his viewing angles and velocity he had at the time he teleported. These inherited view angles can still be modified by the angles value of the destination, if desired. LANDMARK teleporters support the optional use of the **trigger_transition**.

And, **Lazarus movewith** support allows the trigger to move with a parent entity.

Key/value pairs

count

When non-zero, specifies the number of times the teleporter will be touched before it is auto-killtargeted (see [this page](#) for details). Default=0.

movewith

Targetname of the parent entity the trigger_teleporter is to **movewith**.

noise

Specifies the path and filename of the .wav file to play when the teleporter transmits to its destination. If no file is specified, no sound will be played. Ignored if **CUSTOM_SOUND** is not set.

target

Targetname of the entity the teleporter will use as a destination.

targetname

Name of the specific trigger_teleporter.

Spawnflags

TRIGGERED Spawnflag (=2)

The teleporter is inactive when the map loads and must be targeted to activate it.

MONSTER Spawnflag (=8)

The teleporter can transmit monsters and actors.

NO_EFFECTS Spawnflag (=16)

Normal teleporter particle effect and sound are disabled.

CUSTOM_SOUND Spawnflag (=32)

The teleporter will call a custom .wav file when transmitting to the destination. This file is specified with the **noise** key.

LANDMARK Spawnflag (=64)

The teleporter will cause the player's view angles and velocity to be preserved when he spawns at his destination. View angles are rotated by the angles value of the destination entity minus the inherited

angles. If this flag is set, a **trigger_transition** may be used.

NOT_IN_EASY Spawnflag (=256)

The trigger_teleporter will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_teleporter will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_teleporter will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_teleporter will be inhibited and not appear when deathmatch=1.

Lazarus Main Page

Trigger_transition

The trigger_transition is a new **Lazarus** brush model for Quake2 which is inspired by the Half-Life entity. It acts to create more seamless level changes by allowing many entities within its field to change levels along with the player. Yes this means monsters can change levels too. So can a lot of other things... but there are exceptions, which are outlined in [this list](#).

The transition cannot be triggered independently; it is activated along with a [target_changelevel](#), [misc_teleporter](#), or a [trigger_teleporter](#), as long as these entities have the **LANDMARK** spawnflag set. The trigger_transition should have the same targetname as the transporting entity. When the transporting entity is triggered, so will the trigger_transition.

All entities transported within the trigger_transition field will be reoriented as required, as outlined in the LANDMARK spawnflag descriptions you'll find on pages detailing the changelevel and teleporter entities.

This entity works by deleting entities from the map or area the player is leaving and spawning them in the map or area the player is entering. The "new" entities will spawn inheriting the keys and spawnflags, and the current status of their "predecessors", so you can rely on the "new versions" to act the same as the "old" ones. If a monster changes levels with the player, he is removed from the monster count of the map he came from, and is added to the monster count of the map he entered. (This only applies to living monsters, not corpses - although corpses can certainly make the transition).

Many entities **will not** be affected by the trigger_transition, because of the complications that may be created if they were. This includes brush models and path nodes. For best results, the analogous areas in the two maps should be built so that such entities cannot possibly be present within the players view when he changes levels, or else they should be duplicated in both maps. Naturally this means that moving brush models shouldn't be used in the changelevel areas.

Since the trigger_multiple and similar trigger fields are brush models, they will not make the transition. Since the [trigger_bbox](#) is a point entity, it **will** make the transition. We're calling attention to this in the event you have a use for this property, which is unique among all the trigger field entities.

A table listing all entities which **are not** affected by the trigger_transition can be found at the [end](#) of this page.

Further, some entities are "owned" by other entities. This includes a rocket fired by a monster or actor. Using this as an example, if a monster outside of the trigger_transition field fires a rocket which is inside the field when the trigger_transition is activated, the rocket will not change levels because its owner (the monster) didn't change levels. While on the subject of rockets: A homing rocket that changes levels will revert to a standard rocket in the next level.

Another example of an "owned" entity is one that is set to [movewith](#) another entity. If the parent entity is not affected by the trigger_transition (and no parent is), then its movewith children won't be either.

We did have some concerns with the game's autosave feature. Suppose the player inadvertently triggers a level change while in the midst of a battle where his health has dropped to almost nothing. If both he and his enemies make the transition, the player may lose his last health points in the first fraction of a second in the new map. Which would leave the player with an autosaved game which would spawn him in the new map an instant before his death. To avoid this, the code for the [target_changelevel](#) has been modified so that the player will not suffer any loss of health below 10 health points for the first 3 seconds after a new map loads. (This does not affect armor). This gives the player a chance to find health, finish off his enemy, and/or take cover - rather than leave him with a worthless saved game.

Key/value pairs

targetname

Name of the specific trigger_transition. The targetname should be identical to the targetname of the [target_changelevel](#), [misc_teleporter](#), or [trigger_teleporter](#) that it's designed to work with.

Spawnflags

NOT_IN_EASY Spawnflag (=256)

The trigger_transition will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The trigger_transition will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The trigger_transition will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The trigger_transition will be inhibited and not appear when deathmatch=1.

Exempted Entities - which DON'T make transitions

all brush models
crane_reset
func_clock
func_timer
hint_path
info_player_*
light
light_mine1
light_mine2
misc_strogg_ship
misc_viper
misc_viper_bomb
model_train
path_corner
path_track
point_combat
target_actor
target_changelevel
target_character
target_crosslevel_target
target_crosslevel_trigger
target_goal
target_help
target_lightramp
target_locator
target_lock*
target_rotation
target_secret
target_string
trigger_always
trigger_counter
trigger_elevator
trigger_key
trigger_relay
turret_driver
worldspawn



Miscellaneous triggers

trigger_inside

Trigger_inside is similar to a trigger_multiple, but only fires its target(s) when the bounding box for the pathtarget entity is completely inside the trigger field.

trigger_mass

Trigger_mass is similar to a trigger_multiple. It fires its target(s) when touched by **any** entity (not just the player or a monster) whose mass is greater than or equal to the mass value for the trigger. This trigger might be useful for requiring the player to drop or otherwise move a **func_pushable** into position to "dislodge" a stuck lever or break a barrier. The Q2 player mass is 200. Default mass for trigger_mass = 100.



trigger_scales

When teamed with multiple target_characters, trigger_scales can be used to weigh the entity or entities that are "supported" by the trigger field. The weight display **does** include any stacked entities. The weight value displayed is the fraction of the entity's bounding box that is contained by the trigger field. The target_character count values are from right-to-left, in other words count=1 is the 1's place, count=2 is the 10's place, etc. By itself this isn't a particularly useful entity, but is a nice effect for a warehouse-type environment.



Lazarus Main Page

Turn_rider

Stand on a rotating brush in the regular game, and you'll ride it, but you won't turn with it. Drop an item on that rotating brush model and it will go round and round, but won't turn either. This simply looks silly. A more realistic effect would be for the entity riding the rotating bmodel to turn as well, and **Lazarus** offers the option to make this so.

We *could* have changed this globally, so that every single map played under **Lazarus** would see their rotating bmodels exert this effect on entities that ride them, but we decided not to - trying to shoot targets while riding on rotating brushes is quite a bit harder if you're turning with the brush, and we didn't like the idea of changing the effect so drastically for existing maps. What **Lazarus** does offer instead is the **turn_rider** option for rotating brush models. This is a new key with 2 possible values: set it to **0** and the objects riding the bmodel will not turn, set it to **1** and they will. So new maps, and only new maps, can take advantage of this feature, or not, as desired. And, an advantage of using a keyvalue is so this feature can be enabled for some rotating bmodels and not others in the same map.

Turn_rider, when used, affects the player, monsters, items, and anything else that might be riding on the rotating brush. Note that the game "thinks" in 0.1 second intervals, so depending on the distance of the riding entity from the center of the origin of the rotating bmodel, and the speed at which the rotating brush is turning, the effect may possibly seem choppy. As with everything else, specific cases require specific testing.

Func_pushables will be turned by a turn_rider rotating brush, just as point entities are. However, the pushable needs its origin defined - either place it a 0 0 0 (map origin) or incorporate an origin brush as part of the func_pushable.

Another use of turn_rider concerns **movewith** parent-child combos. If the parent entity rotates, you can use the value of turn_rider to specify whether or not the child will see the rotation of the parent applied to it. This is useful for modeling machines and the like. A special case is when turn_rider is a property of a **func_vehicle** - with that entity, the rider (the player) already turns with the vehicle no matter what the setting of turn_rider - what it affects is whether or not the vehicle's movewith children (if any) are to rotate as the vehicle turns.

Turret_base

The **Lazarus turret_base** is nearly identical to the normal Quake2 turret_base, with the exception of the addition of the **TRIGGER_SPAWN** spawnflag. This is to allow your turrets to appear when they are triggered, for simulating a pop-up turret for example (more on this in the [turret_breach](#) page).

The function of the turret_base is to allow its teamed [turret_breach](#) to rotate on the X-Y plane (yaw). A turret_base **always** has its axis of rotation on the Z-axis. Limiting the yaw range is accomplished with the keyvalues assigned to its teamed [turret_breach](#). The turret_base must be placed in the map with its "forward" side facing 0 degrees; its facing direction when spawned in the game is determined by its **angle** keyvalue (which in practice should be the same as the angle value given to its teamed breach).

The original Quake2 turret_base could be made to spawn in the game at some odd angle using the **angles** (pitch/yaw/roll) keyvalue. However, the **turret_breach** ignores the pitch and roll elements, making this ability worthless. For consistency, the **Lazarus** turret_base ignores the pitch and roll elements as well.

Turret_base and [turret_breach](#) can use animated textures, and are animated only when "owned" by either a player or [turret_driver](#).

Key/Value pairs

angle

Direction in degrees you want the turret_base to face when spawned in the game.

movewith

Targetname of the entity to move with.

targetname

Name of the specific turret_base.

team

Value shared by the corresponding [turret_breach](#).

ArghRad bmodel keys

All special brush model lighting effects using the [ArghRad](#) radiosity compiler are supported.

Spawnflags

TRIGGER_SPAWN (Spawnflags=2)

When set the turret_base will not spawn in the game until triggered, by calling its [targetname](#).

[Lazarus Main Page](#)

Turret_breach

The **Lazarus** version of the standard Quake2 turret_breach brush model goes way, way beyond the enforcer standing on the artillery piece. There's lots and lots of things you can do with them now:

- **Player-Controlled Turrets:** Traditional ride-on turrets may be controlled and fired by the player, using the **PLAYER** spawnflag. In a single-player game you can kill the driver and take over his turret. Simply hopping onto the turret at any point *32 units behind the turret origin* puts the player in control, complete with unlimited ammo. *Jumping* causes the player to abandon the turret.
- **No-Driver Turrets:** Turrets can dispense with the **turret_driver** entirely, acting as a Rogue-like (Ground Zero) automated turret. But since they're brush models, you can have variety in design. No-driver turrets need the **TRACK** spawnflag set.
- **Not Just Rockets:** Multiple weapon firing modes are available by setting the **sounds** key. Rate of fire is adjustable by using the **wait** key. Turret firing can also be suppressed entirely, so you can build a security camera that tracks monsters and players.
- **Remote-Control Turrets:** With the addition of a **func_monitor**, the player can control the turret (or camera) remotely. Aiming is accomplished using the normal *freelook* or *turn/pitch* commands. Attacking is done as you would expect, except you have unlimited ammo :). Multiple turrets can be accessed and controlled from the same location, if the turrets share the same **targetname**. Switching from one turret to the next and back again is done using any of the normal *strafing* commands. Turret access order from a single **func_monitor** is determined with its **count** key.
- **Ambush Turrets:** The **TRIGGER_SPAWN** spawnflag allows you to keep a turret from appearing in an area until you want it to be there. Even better, you can have true popup turrets by using the **movewith** key.
- **Roving Turrets:** The **movewith** key can be used to have turrets roam around the room rather than just being fixed in one place. And yes, they'll fire on the player the whole while.
- **Player-Destroyable:** The turret_breach can be made to be "killable" and destroyable by assigning **health** and **gib_health** values. Separate events can be triggered at the moments of both the turret's death and its destruction, with the use of **deathtarget** and **destroytarget** keys.
- **Activity Indicators:** Turret_breach and **turret_base** can use animated textures, and are animated *only* when "owned" by either a player or **turret_driver**. This is useful for creating pulsing indicator lights. See either turret.map or turret2.map in the examples. (You'll need to start a deathmatch or coop game to see this feature working in turret2).
- **Difficulty Varies with Skill Level:** **TRACK** turrets automatically scale their difficulty, based on skill value. As skill gets harder, the turret will fire more frequently (see **wait**). Also, the turret will track your movements more closely on higher skill settings. When skill=0, the turret will follow the target's last position (normal Q2 targeting), when skill=1, it will follow the target's current position, and when skill=2, the turret will lead the target. Fast-turning turrets with instant-hit weapons are pretty deadly on hard skill, but are not impossible to avoid as long as your circle-strafing is up to it.

Player-control of the ride-on turret is straight from MapPack, with minor modification.... thanks to Mr. Ed for making the source code available, but more importantly for getting this thing working much better than the MapPack original.

Turrets are not the easiest entities to make, and are not too often made by mappers for the original Quake2 game. So some tips on turret construction and use are in order. As is some blather on stuff to watch out for when making and using some of the **Lazarus-enhanced** turret types, and for Q2 turrets in general. Said blather can be found at the **end of this page**.

Legend:

Compared to Standard Quake2, Key/Flag is:

Key/value pairs

angle

Specifies the facing angle of the turret on the XY plane. The turret_breach should be constructed so that its intended "front" faces 0; the value of angle determines the direction the front will face when the map is loaded. Default=0.

count

Specifies the position number in a series of turrets targeted by the same func_monitor. Switching from one turret to the next and back again is done using normal strafing controls. Count values for the turrets should be sequential, starting at 1. If the count value for the first turret targeted by a func_monitor in the map is unspecified or 0, then the code will cycle through the cameras in the order they appear in the .bsp file. If a turret in the order is not found, the code will skip to the next turret in the count when cycling.

deathtarget

Trigger which fires when the turret's health is depleted.

destroytarget

Trigger which fires when the turret's gib_health is depleted. **Note:** the code ensures that the destroytarget entity is triggered no less than 0.2 seconds after the turret "dies". This check is performed to ensure that the sequence meant to occur actually takes place. For example, if you set both the deathtarget and destroytarget to a func_timer that controls a target_splash, this check ensures that the gib event will actually toggle the func_timer off again.

distance

Sets the initial velocity of fired grenades in units/sec. This setting will affect the effective range of grenade turrets. If a grenade turret can see its target but cannot hit it, it won't fire. Ignored unless sounds=7. Default=800.

Note: The game limits the velocity of entities (sv_maxvelocity) to 2000, and the code will add some variable lateral velocity to the grenade's initial velocity; therefore distance>1990 will result in an inaccurate GL turret.

dmg

Damage done to the entity blocking the turret's rotation, in health points/0.1 second. Default=10.

followtarget

Targetname of the entity a TRACK turret will aim at and follow, if there are no better targets found.

gib_health

Hit points of an already-disabled turret. When the gib_health is depleted, the destroytarget trigger will fire. If this value is greater than or equal to 0, disabling the turret results in its destruction as well. Default=0.

health

Hit points until disabled, at which point the deathtarget trigger will fire. If health=0, the turret is indestructible. Default=0. **Note:** If the turret has an accompanying turret_driver, his health will be linked to the turret_breach, so that killing the breach will also kill the driver.

maxpitch

Maximum allowable pitch angle. Default=30.

maxyaw

Maximum allowable yaw angle. Default=0.

minpitch

Minimum allowable pitch angle. Default=-30.

minyaw

Minimum allowable yaw angle. Default=360.

movewith

Targetname of the parent entity the turret assembly (turret_breach and turret_base) is to movewith.

sounds

Specifies what the turret will fire. Firing behavior is determined by wait, and in the case of grenade turrets, distance as well. Default=2. Choices are:

- 1: No Fire
- 1: Railgun
- 2: Rockets
- 3: BFG10K
- 4: Homing rockets
- 5: Machinegun
- 6: Hyperblaster
- 7: Grenades

speed

Speed of pitch and yaw rotations in degrees/second. Default=50.

target

Targetname of an info_notnull, so placed as to determine the weapon firing origin distance from the breach's origin. Every turret **must** target an info_notnull in order to function.

targetname

Name of the specific turret_breach. TRACK turrets may be toggled on/off by triggering them.

team

Value shared by the corresponding turret_base. With Lazarus, non-solid teammates will both yaw **and** pitch with the turret_breach. What's this good for? How about attaching a non-solid model_spawn gun model or gun sight or other model to the turret?

viewmessage

Message which appears on the screen when a turret is accessed using a func_monitor, either through initial access or when switching from one turret to the next.

wait

The meaning of "wait" is dependent on the weapon type (sounds) used.

Firing rate: When sounds=1, 2, 3, 4, or 7 (RG, RL, BFG, HomingRL, GL), "wait" sets the number of seconds to wait between weapon firing events. **Note:** An additional skill-level dependent "reaction time" delay is added to this, as described by the formula:

$$[\text{reaction time}] = [2 - \text{skill}] / 2$$

So if a RL turret uses wait=1, on normal skill the actual time between firings would be 1.5 seconds.

Damage level: When sounds=5 or 6 (MG or HB), "wait" sets the amount of damage points each shot will do (firing rate is fixed at 10 shots/second).

Ignored when sounds=-1. Default=2.

Spawnflags

PLAYER Spawnflag (=1)

The turret may function as a "ride-on" turret. **Not** incompatible with turret_drivers; but you do have to kill the driver before the player can take control. Player riding position should be 32 units behind the turret_breach's origin.

TRIGGER_SPAWN Spawnflag (=2)

The turret_breach will not spawn in the game until triggered, by calling its targetname.

TRACK Spawnflag (=4)

The turret will act without being controlled by either a turret_driver or a player. It is fully automated and

will look for the nearest player, monster, corpse, or **followtarget** (in that order) and follow their movements, if any. If **sounds** is **not** equal to -1 (no fire) then a non-**GOODGUY** turret will **always** select the closest player if he is visible and will then ignore other targets. **GOODGUY** turrets ignore players if a monster is visible.

GOODGUY Spawnflag (=8)

Turret will attack monsters rather than players. If set, **TRACK** is automatically turned on also. **NOTE:** To prevent killing monsters at map startup before the player arrives on the scene, **GOODGUY** turrets are initially disabled and must be triggered into activity by calling its **targetname**.

NOT_IN_EASY Spawnflag (=256)

The turret_breach will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The turret_breach will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The turret_breach will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The turret_breach will be inhibited and not appear when deathmatch=1.

Construction tips

To start with, turrets need both a **turret_breach** and **turret_base** to function properly, and they should be **teamed** together so the game associates them. Breaches always rotate around an axis which runs parallel to the X-Y plane. For the game to know which end of the turret is the business end, it must be placed in the map so that end faces **0 degrees**. Facing angle when spawned in the game is determined by the **angle** key. So, make both your breach and base face 0 degrees in your map, and give them both the same angle value. Determining the initial facing **pitch** angle is not possible, but the **Lazarus** turret can be made to look wherever you want it to by using the **followtarget** key to point it there.

For best visual and performance results, the axis of the breach and the axis of the base should **intersect**. In other words, make sure a vertical line drawn through the center of the base's origin brush intersects with a horizontal line drawn through the center of the breach's origin brush. The two origin brushes themselves do not need to intersect.

The turret_breach **must** **target** an **info_notnull**. The distance of the info_notnull from the breach's origin determines the distance from the origin that the weapon projectile will originate. If the turret doesn't fire as it should, the info_notnull may be too close to the breach.

For info on the **turret_driver** element in all this, refer to the, err, **turret_driver** page.

Turrets in operation

Once you have your turret functioning, there are still a number of ways to slip up.

PLAYER ride-on turrets should be thoroughly tested so that the player doesn't get crushed as the breach pitches up and down. Adding a clip brush or two to the breach and/or the base can turn the trick here without compromising the look of your turret.

Make sure that turrets can't get fouled up in nearby world geometry, by making them smaller/shorter, moving them to a clear area, limiting their pitch/yaw, and/or moving the offending world brushes out of the way.

Another consideration that applies to all turrets concerns an oddity when attempting to limit their allowable yaw (**minyaw/maxyaw**). A turret will always turn the least number of degrees to find its target. If you limit your turret to a 270-degree yaw range, it will often pass through its "forbidden" arc to get to where it needs to go if that's the shortest way to get there. This looks silly and should be avoided; if you try this you'll see why. Therefore, a yaw range of anything greater than 179 degrees is not really a good idea.

There are a couple of very special considerations here concerning **Lazarus TRACK** turrets that can be controlled

remotely using a `func_monitor`:

- The *player's viewpoint is determined by the placement of the info_notnull*. While the standard Q2's turret could have its `info_notnull` theoretically placed anywhere on the locus of points described by the surface of a sphere using the breach's origin as a center, this is not true of the **Lazarus** turret. Care must be taken to place the `info_notnull` directly in front of the breach's muzzle as it is placed in the map. Failure to do this will probably result in the player's viewpoint being the breach's origin instead, and the breach may act oddly.
- In the game, the player's min/max pitch (lookup/lookdown) is +89/-89 degrees. It's probably wise to make remote player-controlled turrets adhere to these limitations as well when assigning `minpitch`/`maxpitch` values.

Turrets and Triggers

Many triggers (such as the `trigger_multiple` and `trigger_look`) can be made to detect the "presence" of the player at the turret when he is operating a **TRACK** turret through a `func_monitor`. You could make a scripted event take place only when the player looks through the turret camera, or only when he looks at a certain place. However to make this work properly, you have to realize that when the player is in control of such a turret, his location (for trigger detection purposes) is at the turret muzzle. Which is where you place the turret_breach-**targeted** `info_notnull`. In-game, that position moves as the turret rotates; so make sure the trigger field encloses the entire locus of points that the muzzle may occupy when the turret moves.

More Background

It's probably helpful to understand how **Lazarus TRACK** turrets operate so that you can use them to their best advantage. A **TRACK** turret spawns in the game facing the direction specified by its `angle` value just as any other turret. If this is a weapon-firing (`sounds>0`) turret, it looks first for a player, and fires at him. If there are no players available, it looks for a monster and tracks him, but does not fire. Corpses are included in this check. No-fire (`sounds=-1`) turrets do not prioritize players as targets. Multiple targets are handled by targeting the closest one. The turret will continue to track its current target until there is another target that is 100 units closer. If neither players nor monsters are available, the turret will track the entity specified by its `followtarget`. If no `followtarget` is specified then the turret will stay facing the way it faced last.

When determining the viability of a target, the **TRACK** turret first considers all targets in its range of visibility (as determined by the **PVS table**), then discards all targets that fall outside its allowable yaw range (as determined by its `minyaw`/`maxyaw` values), and then discards all targets that it does not have direct line-of-sight on. What's left is fair game.

These calculations come at a processing cost of course. Depending on what else is going on in a map at the same time, you'll probably notice no slowdowns with a handful of **TRACK** turrets, but a dozen will probably be another story. No hard and fast rules here; every map is different and the visibility each of your turrets would have to deal with varies as well. Use discretion when filling a map up with turrets. If you want to use a lot of them in a large, busy map, then consider requiring the player to destroy turrets as he goes, and use **TRIGGER_SPAWN** turrets for those that will appear later, so only a portion of the turrets are active and "thinking" at any time.

Turret_driver

The **Lazarus turret_driver** makes minimal changes that can have a dramatic effect, and also adds much more versatility to the standard just-standing-there dopey driver. When mated to a traditional **turret_breach** which has been given its own **health** value, the driver's health becomes linked to that of his turret. This is to prevent the occurrence of turret-less drivers. When the turret is killed, so its driver will be too. If the turret has the **PLAYER** spawnflag set, the player can take control of the turret after killing the driver.

Much more significant is the addition of the **REMOTE** spawnflag. When this is enabled, the turret_driver can control a turret remotely. While the driver can always be killed as before, there are two conditions that will cause the **REMOTE** driver to **convert into a regular monster_infantry**. The first is if his turret is killed, and the second is if the driver takes damage. This is so the player can't just come up behind the driver and nibble away at his health with a cheap weapon without fear of reprisal.

A **REMOTE** turret_driver's viewpoint is from the info_notnull of the **turret_breach** he controls. This means it isn't necessary for the driver to see the player before he can fire upon him; only his turret needs to. Also, for the sake of self-preservation, a driver-controlled turret will not fire at points within 128 units of its driver.

When a turret_driver is in control of a turret, that **turret_breach** and **turret_base** team will have active animated textures, if applied. If the driver is killed, or gives up control of the turret, the textures will no longer animate.

Key/Value pairs

angle

Direction in degrees you want the turret_driver to face.

angles

Used instead of **angle**, separate pitch, yaw and roll elements can be specified.

target

Targetname of the **turret_breach** controlled.

targetname

Name of the specific turret_driver.

Spawnflags

REMOTE (Spawnflags=1)

When set the turret_driver can control a turret remotely, and under certain circumstances may transform into a monster_infantry.

Lazarus Main Page

Weapon_*



Lazarus adds a few notable enhancements to all weapon_* point entities, which can allow for their more realistic placement in the map as well as making them destroyable by weapon fire. And, **movewith** support is also included. The entities affected are:

- **weapon_blaster**
- **weapon_shotgun**
- **weapon_supershotgun**
- **weapon_machinegun**
- **weapon_chaingun**
- **weapon_grenadelauncher**
- **weapon_rocketlauncher**
- **weapon_hyperblaster**
- **weapon_railgun**
- **weapon_bfg**

Please see the descriptions for the **NO_DROPTOFLOOR**, **NO_STUPID_SPINNING** and **SHOOTABLE** spawnflags for more details.

As a kind of a side note, you can also specify what weapon (if any) that the player will start out with when a map is first loaded. For the scoop on how to do that, see the **worldspawn** page.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

angle

Specifies a facing direction on the XY plane. Default=0. Ignored unless **NO_STUPID_SPINNING** is set.

angles

Specifies a facing direction in 3 dimensions, defined by pitch, yaw, and roll. Default=0 0 0. Ignored unless **NO_STUPID_SPINNING** is set.

delay

Specifies the delay in seconds before **target**, **killtarget** and **message** will fire after the weapon is touched by the player. Default=0.

health

Specifies hit points before the weapon will explode if the **SHOOTABLE** spawnflag is set. Default=20.

killtarget

Targetname of the entity to be removed from the map when the weapon is touched by the player.

message

Specifies the character string to print to the screen when the weapon is touched by the player.

movewith

Targetname of the parent entity the weapon is to **movewith**.

target

Targetname of the entity to be triggered when the weapon is touched by the player.

targetname

Name of the specific weapon entity.

team

Team name of the specific weapon entity. This is only relevant when deathmatch=1. When multiple pickups have identical team names, the first to appear in the .map file will be the one that appears on map load. This pickup entity serves as the master, and its teammates are the slaves. When the master pickup entity is picked up by a player, any of the teammates will spawn on a randomly rotating basis. Weapon entities may be teamed with other weapons, **ammo**, and/or **items**.

Spawnflags

NO_STUPID_SPINNING Spawnflag (=4)

The weapon_* entity will not spin. This allows proper use of the **angle** and **angles** keys. When used in conjunction with the **NO_DROPTOFLOOR** spawnflag, weapons may be placed in a realistic manner in your map.

NO_DROPTOFLOOR Spawnflag (=8)

The weapon_* entity will remain wherever it was placed in the map editor. This allows for placing the entity where it could be left hanging in the air, and/or it could be placed so that its bounding box intersects any solid brush, for example snugging the weapon up against a wall or laying it on its side. The normal 32x32x32 pickup "field" would still be in effect, so care must be taken so that the situation where the weapon can be picked up from the opposite side of a wall or similar is avoided. The weapon will not be lit if the entity's origin is buried in a world brush or otherwise hidden from light. The runtime "DropToFloor:" error diagnostics will not apply if this spawnflag is set. Please see **these tips** on making use of this feature easier.

SHOOTABLE Spawnflag (=16)

The weapon_* entity will be destroyable by weapon fire, as determined by **health**. The downside to using this option is that the weapon must be made solid. In the game, there will be a barely noticeable bump as you run into the model. When deathmatch=1, a destroyed weapon entity will respawn in 30 seconds.

NOT_IN_EASY Spawnflag (=256)

The weapon_* entity will be inhibited and not appear when skill=0.

NOT_IN_NORMAL Spawnflag (=512)

The weapon_* entity will be inhibited and not appear when skill=1.

NOT_IN_HARD Spawnflag (=1024)

The weapon_* entity will be inhibited and not appear when skill=2 or greater.

NOT_IN_DEATHMATCH Spawnflag (=2048)

The weapon_* entity will be inhibited and not appear when deathmatch=1.

[Lazarus Main Page](#)

Weapon_blaster

Lazarus offers the ability to place the weapon_blaster point entity pickup in your map, just as you would any other weapon pickup. You may well ask why anyone would want to do this, since the player starts out with the blaster, and it needs no ammo - what's the point of extra blasters? The reason is simple: if you check out the [worldspawn](#) page, you'll see that you can specify what weapon (if any) that the player will start out with when your map is first loaded. If the player doesn't start out with the blaster, well then a blaster pickup he could find later might come in handy.

Please refer to the [Weapon_*](#) page for a full list of all keys and spawnflags which can be used by the weapon_blaster.

[Lazarus Main Page](#)

Worldspawn

Worldspawn properties allow for global map settings. **Lazarus** offers a couple of options over and above that of plain-vanilla Quake2.

The **style** setting lets you specify which weapon (if any) that you wish the player to start out with when the map is first loaded. The standard-issue blaster can be given along with another weapon, or not, as you prefer. This property is available to **Lazarus** player spawn points as well, and will *only* be used if not specified by the spawn point itself.

The **effects** value allows you to change basic behavior of game components and/or add new features.

The **fogclip** worldspawn property has a somewhat involved explanation... let's say you'd like to create a fogout visibility effect (fog clipping) so as to get around Quake2's annoying maximum visibility distance. The problem that arises is that what you intend the player to see might not be what he actually sees depending on whether he's using the OpenGL renderer or the 3dfx miniGL. This can be corrected by a "proper" setting for "gl_clear", dependent on the current value of "gl_driver". However, overriding the user's video settings is not something one should be too sanguine about. That's where **fogclip**, a **Lazarus** worldspawn property, comes into play. When enabled, "gl_clear" is forced to the setting that will allow for fog clipping without visual anomalies. When disabled, the user's "gl_clear" setting is left alone. This mechanism keeps the monkeying with the user's "gl_clear" setting limited to maps that are really using fog clipping. When the user loads another map (even if he quits first), his normal "gl_clear" setting will be used. Bottom line on **fogclip**: Use it (fogclip=1) only for maps that use fog clipping.

Attenuation and **shift** values are used to control global **target_playback** settings for 3D sounds.

Legend:

Compared to Standard Quake2, Key/Flag is:

New	Modified	Unchanged
-----	----------	-----------

Key/value pairs

attenuation

Sets the global attenuation rolloff factor for target_playback **3D** sounds. Setting this value makes the sound drop off faster or slower. The higher the value, the faster volume will fall off. The lower the value, the slower it will fall off. Default value = 1.0.

bleft

This value is ignored if the CUSTOMHELP **effects** flag is not set. The first two numbers specify the upper left corner of the picture displayed when the "cmd help" button (normally F1) is pressed. The third number is ignored. The actual placement of the picture is resolution-dependent. (0,0) is **not** the upper-left corner of the screen, unless you happen to be playing with a 320x240 display. Standard help computer is placed at (32,8). You can use negative values, but then you risk having the pic being clipped on low-resolution displays. On a 640x480 display, (-160,-120) is the upper-left corner of the screen. If you make a decision that "nobody in their right mind plays Q2 at less than 640x480", which is probably a valid statement, then placing the picture so that it fits in the upper-left corner of a 640x480 screen is likely OK.

effects

Effects flags are used to change the behavior of the game. These values may be combined. Currently valid effects values are:

1: CUSTOMHELP

If set, the normal help computer statistics are not displayed, but pics/help.pcx is (presumably you'd want to use a custom pcx for this purpose). If CUSTOMHELP is set then the position of the picture on the screen may be controlled with the **bleft** setting. Note that as with all Q2 images, the maximum size of the picture is 131072 total pixels. Another consideration is that if the picture is larger than 320x240 then parts of the picture will be clipped on a 320x240 display.

2: STEPSOUNDS

If set, the game will play sloshing sounds when the player moves through shallow water, wading sounds when the player moves through slightly deeper water, and another set of custom sounds when the player climbs a ladder. We don't play those sounds by default because the addition of the 3 stock wading sounds might break existing maps with ERROR: Index Overflow. (Other footstep sounds use FMOD, bypassing the Q2 sound system.) **NOTE:** Wading sounds are stock Q2 files; others are custom sounds that must be redistributed with your map if used. For more information check out the [footsteps](#) documentation.

4: WHATSIT

If set, then the common names of monsters and pickup items that the player is looking at are displayed immediately above the normal statusbar. This feature may eventually be expanded to include a bit more information.

8: ALERTSOUNDS

If set, monsters are alerted by player footstep and falling sounds. Use this flag if you want to reward stealth and/or penalize carelessness.

16: CORPSEFADE

If set, monster, actor, and misc_insane corpses begin to fade away and sink into the floor [corpse_fadetime](#) seconds after death. Aside from looking cool, this feature will clean up exceptionally active maps and help prevent overflow errors. Monster_medics are removed from the game if this flag is set.

32: JUMPKICK

If set, player can damage monsters, actors, func_explosives, and other damageable entities by jumping into them as in Action Quake2. Jump-kick doles out 10 damage points to the recipient, and knockback is proportional to the mass of the kicke. You can force this feature on by setting the [jump_kick](#) cvar to 1.

fogclip

Specifies whether to force the user's "gl_clear" setting to an appropriate value for the proper rendering of fog clipping effects. If fogclip=1, and the user's setting for gl_driver=3dfxgl, gl_clear is forced to 0. If fogclip=1, and the user's setting for gl_driver is anything else, gl_clear is forced to 1. If fogclip=0, gl_clear is left untouched. These settings are necessary to draw the correct background color to obscure distance clipping and HOM effects when using fog, as described in the [Fog](#) documentation. Choices are:

0: Leave the user's gl_clear setting alone (default).

1: Force gl_clear to a setting appropriate for his GL renderer.

gravity

Specifies the gravity level. Default=800.

message

Specifies the title of the map.

nextmap

Specifies the name of the next map to be loaded when timelimit or fraglimit is reached; ignored when deathmatch=0.

shift

Sets the doppler shift scale factor for target_playback [3D](#) sounds. This is a general scaling factor for how much the pitch varies due to doppler shifting. Increasing the value above 1.0 exaggerates the effect, whereas lowering it reduces the effect. Use a value less than 0 to remove the effect altogether (recommended for 3D music). FMOD's speed of sound at a shift value of 1.0 is roughly 13600 units/second. Default value = 1.0.

sky

Specifies the name of the environment map to be used.

skyaxis

Specifies the axis the sky will rotate around. Ignored if [skyrotate](#) is not set.

skyrotate

Sets the speed of a rotating sky in degrees/second.

sounds

Specifies the CD track to be played.

style

Specifies the player's starting weapon, unless overridden by a style setting used by a spawn point (`info_player_start`, etc). Ignored when the map is loaded as a result of triggering a `target_changelevel`.

Choices are:

- 0:** Don't specify (default)
- 1:** Blaster
- 2:** Shotgun, Blaster backup
- 3:** Super Shotgun, Blaster backup
- 4:** Machinegun, Blaster backup
- 5:** Chaingun, Blaster backup
- 6:** Grenade Launcher, Blaster backup
- 7:** Rocket Launcher, Blaster backup
- 8:** Hyperblaster, Blaster backup
- 9:** Railgun, Blaster backup
- 10:** BFG10K, Blaster backup
- 1:** No weapon at all
- 2:** Shotgun, no Blaster
- 3:** Super Shotgun, no Blaster
- 4:** Machinegun, no Blaster
- 5:** Chaingun, no Blaster
- 6:** Grenade Launcher, no Blaster
- 7:** Rocket Launcher, no Blaster
- 8:** Hyperblaster, no Blaster
- 9:** Railgun, no Blaster
- 10:** BFG10K, no Blaster