Extended General Line and Sector Type Reference

Version 1.0
by Jaakko Keränen <skyjake@doomsdayhq.com>
October 13, 2002

1 Introduction

This document describes what Extended General Line and Sector Types (XG lines and sectors) are and what they can do. It is meant for Doom (Heretic, Hexen) level designers who'd like to increase the functionality of their maps without resorting to more complicated solutions like scripts.

1.1 What is XG?

Doom has a number of hard-coded line and sector types, each of which are programmed to do certain effects like lower a lift, open a door or, in the sectors' case, perhaps just to blink randomly. They were designed to be used in the game's own maps and thus do not provide the best possible selection of effects for your ones.

XG line types separate the activation method, requirements and the effects of the line. It is possible to create 'chains' of line types, which effectively combine many types of line into one. Chains are a very important part of XG lines and sectors. Often it is easiest to define several line types, each for a certain function, and then chain them up behind a single type, which is then used in the map.

XG sector types define the physical characteristics of a sector, like light level and color, wind and gravity, and a number of chains that are activated under certain circumstances. Like with lines, sector chains refer to line types. This means all the things that are possible to do by changing the state of lines are also possible to do with sectors.

XG line and sector types are entirely separate from the original Doom types. Both the extended and the standard types can be used in a map.

1.2 Requirements

You need a version of Doom (Heretic, Hexen) which supports the XG specification as described in this document. At the moment the only such version is my Doom port, jDoom (http://jdoom.newdoom.com/) version 1.1 or newer.

1.3 Defining XG data

With the Doomsday engine XG line and sector types are defined with DED files. Once loaded, the XG definitions can be dumped into a binary file using the console command "dumpxg". The file can then be merged into a WAD with the lump name of DDXGDATA. This is recommended for people who distribute their maps that contain XG data. The information in the DDXGDATA lump overrides that in the DED files.

2 Line types

An XG line is a normal Doom line whose type is set to an XG line type. XG lines can either be active or inactive. It is possible to disable an XG line, which means the line will be skipped in all event processing and the line's timer will not be incremented.

Line types are divided into various classes, which define what the line does. The function of a line is executed either when the line is activated or deactivated (or both) or by some other means, which we will go into a bit later on. Some examples of line classes include Plane Move (`ltc_plane_move`), End Level (`ltc_end_level`) and Wall Texture Change (`ltc_wall_texture`). Adding new line classes won't break backwards compatibility.

An XG line type definition begins like this:

```
Line Type {
   ID = 5000;        # A unique ID number.
```

A unique ID number is required for each line type. It is the actual type number that will be used in a level editor.

2.1 Activation and deactivation

A line can be activated and deactivated in various ways. They are based on line events, which occur under certain circumstances. In most cases events have an activator, which is the thing (map object; mobj) that triggered the event. These kinds of events include Use, Shoot, Cross and Touch events, which will take place when a thing uses, shoots, crosses or touches the line (collides with it). There are also Ticker and Chain events. The former is sent by the game ticker (up to 35 times per second) and the latter by a line or a sector. Events will activate inactive lines and deactivate active ones.

A line type can specify several requirements that must be met for the activation or deactivation of the line to succeed. These include for instance event type, a counter (as in "line can be activated N times"), activator type (player, missile, etc.) and any combination of keys possessed by the activator.

Many things can happen when a line is activated or deactivated. The most important one is the execution of the line's function. Other things include showing of messages, playing sounds, changing the texture of the line (handy for switches) and sending chain events.

The activation type of a line specifies how the line behaves. DED definitions of the five activation types available are listed below.

```
Flag { ID = "lat_timed_off"; }
Flag { ID = "lat_timed_on"; Value = 0x1; }
Flag { ID = "lat_flip"; Value = 0x2; }
Flag { ID = "lat_flip_timed_off"; Value = 0x3; }
Flag { ID = "lat_flip_timed_on"; Value = 0x4; }
```

2.1.1 lat_timed_off

Line can be activated if it's inactive. Once activated, it will remain in that state for a given period of time, after which it will automatically deactivate itself. Line can't be deactivated with events.

2.1.2 lat_timed_on

Line can be deactivated if it's active. Once deactivated, it will remain in that state for a given period of time, after which it will automatically activate itself. Line can't be activated with events (i.e. this is the reverse of `lat_timed_off`).

2.1.3 lat_flip

Line can be activated and deactivated with events at any time. There is no automatical time-based activation or deactivation.

2.1.4 lat_flip_timed_off

Line can be activated and deactivated with events at any time. Once activated, the line will deactivate itself after a given period of time.

2.1.5 lat_flip_timed_on

Line can be activated and deactivated with events at any time. Once deactivated, the line will activate itself after a given period of time.

2.2 Chains

Line types can be chained. This means different line types can be combined. This is necessary for some effects that require more than one line class at the same time. A line can have three chains: activation, deactivation and event chain. Chains are always line type IDs. As an example let's say that a line has the activation chain 5120. When the line is activated, it will send a Chain event to itself before its class's function gets executed. During the processing of the Chain event the line is treated just like it were of type 5120.

Event chains work a bit differently. If a line has an event chain, any events sent to the line will first be processed treating the line as it were of the type specified by the chain. If the event passes (it meets the requirements of the chained type) further processing won't be done. In effect this allows having alternate functions and activation methods and requirements for a line.

When Chain events are processed, their activator is the same as the activator of the event that sent the Chain (or in other words, the most recent activator associated with the line).

A door that closes automatically is a good example of where chains are needed. The door line type's activation chain would move the ceiling of the tagged sector to some reasonable height, like to the lowest adjacent ceiling height with an offset of –4. The deactivation chain could alternatively move the ceiling back to its original height or move it to the same height with the floor of the sector, thus closing the door.

2.3 References

In order to provide extended functionality, lines, planes (floors and ceilings) and sectors need to refer to each other in some way. With standard Doom line types this is typically done

with tag numbers, so that the lines and sectors sharing the same tag will participate in the execution of a line's function.

All references except sector-plane and light level references are composed of two parts: a type and a data component. Some types of references have no use for the data component. Sector-plane and light level references only include the type.

2.3.1 Line references (lref)

Line references or lrefs are actually line-line references. They are used when a line needs to refer to a line or a group of lines. DED definitions of the lref types are listed below.

```
Flag { ID = "lref_self"; }
Flag { ID = "lref_tagged"; Value = 0x1; }
Flag { ID = "lref_line_tagged"; Value = 0x2; }
Flag { ID = "lref_act_tagged"; Value = 0x3; }
Flag { ID = "lref_index"; Value = 0x4; }
Flag { ID = "lref_all"; Value = 0x5; }
```

2.3.1.1 lref_self

Refers to the line itself. The data component is not used.

2.3.1.2 lref_tagged

Refers to all lines that have the tag number specified in the data component of the reference.

2.3.1.3 lref_line_tagged

Refers to all lines that have the same tag number as the referring line. If the data component is non-zero, the referring line itself is excluded. Otherwise it is included in the group of referenced lines.

2.3.1.4 lref_act_tagged

Refers to all XG lines whose type's activation tag (Act tag) matches the data component of the reference. This makes it possible to refer to a set of XG lines of one or more specific types.

2.3.1.5 lref_index

Refers to the one specific line whose index number is specified in the data component. Line index numbers are shown in most map editors.

2.3.1.6 lref_all

Refers to all lines in the map. The data component is not used.

2.3.2 Line-plane references (lpref)

Line-plane references or lprefs are used when lines need to refer to planes. Each sector has two planes: the floor and the ceiling. DED definitions of the lpref types are listed below.

```
Flag { ID = "lpref_none"; }
Flag { ID = "lpref_my_floor"; Value = 0x1; }
Flag { ID = "lpref_tagged_floors"; Value = 0x2; }
Flag { ID = "lpref_line_tagged_floors"; Value = 0x3; }
Flag { ID = "lpref_act_tagged_floors"; Value = 0x4; }
```

```
Flag { ID = “lpref_index_floor”; Value = 0x5; }
Flag { ID = “lpref_all_floors”; Value = 0x6; }
Flag { ID = “lpref_my_ceiling”; Value = 0x7; }
Flag { ID = “lpref_tagged_ceilings”; Value = 0x8; }
Flag { ID = “lpref_line_tagged_ceilings”; Value = 0x9; }
Flag { ID = “lpref_act_tagged_ceilings”; Value = 0xA; }
Flag { ID = “lpref_index_ceiling”; Value = 0xB; }
Flag { ID = “lpref_all_ceilings”; Value = 0xC; }
Flag { ID = “lpref_special”; Value = 0xD; }
```

2.3.2.1 lpref_none

Reference to nothing. With some line classes this type of reference has a special meaning. (See Classes)

2.3.2.2 lpref_my_floor

Refers to the referring line’s sector’s floor plane. The sector in question must be on the front side of the line, i.e. the line must be facing the sector. The data component is not used.

2.3.2.3 lpref_tagged_floors

Refers to the floor planes of all the sectors whose tag number is the same as the data component of the reference.

2.3.2.4 lpref_line_tagged_floors

Refers to the floor planes of all the sectors whose tag number matches the referring line’s tag number. The data component is not used.

2.3.2.5 lpref_act_tagged_floors

Refers to the floor planes of all the XG sectors whose type’s activation tag (Act tag) matches the data component. This makes it possible to refer to a set of XG sector floors that belong to sectors of one or more specific types.

2.3.2.6 lpref_index_floor

Refers to the floor plane of the one specific sector whose index number is specified in the data component. Sector index numbers are shown in most map editors.

2.3.2.7 lpref_all_floors

Refers to the floor planes of all the sectors in the map. The data component is not used.

2.3.2.8 lpref_my_ceiling

Refers to the referring line’s sector’s ceiling plane. The sector in question must be on the front side of the line, i.e. the line must be facing the sector. The data component is not used.

2.3.2.9 lpref_tagged_ceilings

Refers to the ceiling planes of all the sectors whose tag number is the same as the data component of the reference.

2.3.2.10 lpref_line_tagged_ceilings

Refers to the ceiling planes of all the sectors whose tag number matches the referring line’s tag number. The data component is not used.

2.3.2.11 lpref_act_tagged_ceilings

Refers to the ceiling planes of all the XG sectors whose type's activation tag (Act tag) matches the data component. This makes it possible to refer to a set of XG sector ceilings that belong to sectors of one or more specific types.

2.3.2.12 lpref_index_ceiling

Refers to the ceiling plane of the one specific sector whose index number is specified in the data component. Sector index numbers are shown in most map editors.

2.3.2.13 lpref_all_ceilings

Refers to the ceiling planes of all the sectors in the map. The data component is not used.

2.3.2.14 lpref_special

Has a special meaning that depends on the context. Similar to `lpref_none` but is treated as a reference even if it really isn't.

2.3.3 Line-sector references (lsref)

Line-sector references or lsrefs are used when lines need to refer to sectors. Lsrefs are basically the same thing as lprefs, but the target of the reference is a sector – not a plane of the sector. See the line-plane references above for the description of the lsref types.

```
Flag { ID = "lsref_my"; Value = 0x1; }
Flag { ID = "lsref_tagged"; Value = 0x2; }
Flag { ID = "lsref_line_tagged"; Value = 0x3; }
Flag { ID = "lsref_act_tagged"; Value = 0x4; }
Flag { ID = "lsref_index"; Value = 0x5; }
Flag { ID = "lsref_all"; Value = 0x6; }
```

2.3.4 Sector-plane references (spref)

Sector-plane references or sprefs are used when lines refer to the features of planes in special or comparative ways. Sprefs are most often used to refer to the height (Z coordinate) or the texture of a plane. When using sprefs a set of sectors or planes has usually already been targeted. Sprefs do not have a data component. DED definitions of the spref types are listed below.

```
Flag { ID = "spref_none"; }
Flag { ID = "spref_my_floor"; Value = 0x1; }
Flag { ID = "spref_my_ceiling"; Value = 0x2; }
Flag { ID = "spref_original_floor"; Value = 0x3; }
Flag { ID = "spref_original_ceiling"; Value = 0x4; }
Flag { ID = "spref_current_floor"; Value = 0x5; }
Flag { ID = "spref_current_ceiling"; Value = 0x6; }
Flag { ID = "spref_highest_floor"; Value = 0x7; }
Flag { ID = "spref_highest_ceiling"; Value = 0x8; }
Flag { ID = "spref_lowest_floor"; Value = 0x9; }
Flag { ID = "spref_lowest_ceiling"; Value = 0xA; }
Flag { ID = "spref_next_highest_floor"; Value = 0xB; }
Flag { ID = "spref_next_highest_ceiling"; Value = 0xC; }
Flag { ID = "spref_next_lowest_floor"; Value = 0xD; }
Flag { ID = "spref_next_lowest_ceiling"; Value = 0xE; }
Flag { ID = "spref_min_bottom_texture"; Value = 0xF; }
Flag { ID = "spref_min_mid_texture"; Value = 0x10; }
Flag { ID = "spref_min_top_texture"; Value = 0x11; }
```

```
Flag { ID = "spref_max_bottom_texture"; Value = 0x12; }
Flag { ID = "spref_max_mid_texture"; Value = 0x13; }
Flag { ID = "spref_max_top_texture"; Value = 0x14; }
```

2.3.4.1 spref_none

Reference to nothing. With some line classes this type of reference has a special meaning. (See Classes)

2.3.4.2 spref_my_floor/ceiling

Refers to the referring line's sector's floor/ceiling plane. The sector in question must be on the front side of the line, i.e. the line must be facing the sector.

2.3.4.3 spref_original_floor/ceiling

Refers to the original floor/ceiling height of a sector. The original height of a plane is set when the map is loaded or after the plane finishes moving (providing the `pmf_setorig` flag is being used by the plane mover).

2.3.4.4 spref_current_floor/ceiling

Refers to the current floor/ceiling height or texture of a sector.

2.3.4.5 spref_highest_floor/ceiling

Refers to the height or texture of the highest floor/ceiling plane of the adjoining sectors.

2.3.4.6 spref_lowest_floor/ceiling

Refers to the height or texture of the lowest floor/ceiling plane of the adjoining sectors.

2.3.4.7 spref_next_highest_floor/ceiling

Refers to the height or texture of the next highest floor/ceiling plane of the adjoining sectors when compared to a sector's current floor/ceiling height (respectively).

2.3.4.8 spref_next_lowest_floor/ceiling

Refers to the height or texture of the next lowest floor/ceiling plane of the adjoining sectors when compared to a sector's current floor/ceiling height (respectively).

2.3.4.9 spref_min/max_bottom_texture

Refers to the specific Z coordinate of the height that is calculated by finding the lower texture with the smallest/largest height (Y) of a sector's all lines and then adding this texture height to the lowest contacted floor height (Z coordinate).

2.3.4.10 spref_min/max_mid_texture

Refers to the specific Z coordinate of the height that is calculated by finding the middle texture with the smallest/largest height (Y) of a sector's all lines and then adding this texture height to the highest contacted floor height (Z coordinate).

2.3.4.11 spref_min/max_top_texture

Refers to the specific Z coordinate of the height that is calculated by finding the upper texture with the smallest/largest height (Y) of a sector's all lines and then subtracting this texture height from the highest contacted ceiling height (Z coordinate).

2.3.5 Light level references (lightref)

Light level references or lightrefs are used when lines need to refer to the light levels of sectors. When using lightrefs a set of sectors or planes has usually already been targeted. Lightrefs do not have a data component. DED definitions of the lightref types are listed below.

```
Flag { ID = "lightref_none"; }
Flag { ID = "lightref_my"; Value = 0x1; }
Flag { ID = "lightref_original"; Value = 0x2; }
Flag { ID = "lightref_current"; Value = 0x3; }
Flag { ID = "lightref_highest"; Value = 0x4; }
Flag { ID = "lightref_lowest"; Value = 0x5; }
Flag { ID = "lightref_next_highest"; Value = 0x6; }
Flag { ID = "lightref_next_lowest"; Value = 0x7; }
```

2.3.5.1 lightref_none

Refers to nothing. Has a special meaning depending on the context.

2.3.5.2 lightref_my

Refers to the referring line's sector's light level. The sector in question must be on the front side of the line, i.e. the line must be facing the sector.

2.3.5.3 lightref_original

Refers to the original light level of a sector. The original light levels are set when the map is loaded.

2.3.5.4 lightref_current

Refers to the current light level of a sector.

2.3.5.5 lightref_highest

Refers to the highest light level among the adjoining sectors.

2.3.5.6 lightref_lowest

Refers to the lowest light level among the adjoining sectors.

2.3.5.7 lightref_next_highest

Refers to the next highest light level among the adjoining sectors when compared to a sector's current light level.

2.3.5.8 lighref_next_lowest

Refers to the next lowest light level among the adjoining sectors when compared to a sector's current light level.

2.4 The Line Type definition

Below you'll find the description for each value in a DED Line Type definition. Values of type "flags" are strings that will be evaluated to an integer when the DED file is read.

2.4.1 ID

Type: integer

A unique integer number identifying this line type. The value is used in level editors to refer to this type. Note that XG line types take precedence, so if the ID number is the same as that of a normal Doom line type, the XG version will be used. IDs must be in range from 1 to 65535.

2.4.2 Comment

Type: string

The Comment property was added to make it easier to manage line types. DED Manager will show this comment in the lookup dropdown lists and in the information section of the main index, but Doomsday itself does not use this string for anything. It's a good idea to make the Comment a short description of what the line type does, for instance:

```
Comment = "A/Lower LT ceil to next lowest";
```

This would suggest that when activated the line will lower the ceiling of the sectors with a matching tag number to their next lowest adjacent ceiling.

2.4.3 Flags

Type: flags

There are several flags that are used to control various aspects about the line type, including activation method and requirements and the time to execute the line's function. The Flags property uses the flag definitions that begin with `ltf_`.

2.4.3.1 Initial state

`ltf_active`	0x1	The line is initially active. If this flag is not set, the line is inactive after the map has been loaded.

2.4.3.2 Activation method

The flags listed below are used to define the type of events that can activate and deactivate a line. Combining the flags is allowed so lines can be activated or deactivated by several kinds of events. The line can also be set to be activated with one type of event and deactivated with another one. There are two versions of each flag, suffixed _a and _d. Using the former activates the line (providing it's inactive) when the event in question occurs. The latter does the opposite. The flag with no suffix contains both the _a and _d versions.

Chain events aren't affected by these flags.

`ltf_player_use`	a:0x2 d:0x4000	Use events from a player mobj can activate/deactivate the line.
`ltf_other_use`	a:0x4 d:0x8000	Use events from a mobj that is not a player can activate/deactivate the line.
`ltf_player_shoot`	a:0x8 d:0x10000	Shoot events (caused by impact weapons: fist, chainsaw, pistol, shotguns, chaingun) whose originator is a player can activate/deactivate the line.
`ltf_other_shoot`	a:0x10 d:0x20000	Shoot events (caused by impact weapons: fist, chainsaw, pistol, shotguns, chaingun) whose originator is not a player mobj can activate/deactivate the line.
`ltf_any_cross`	a:0x20	Cross events (mobj origin crosses the line while moving)

	d:0x40000	from any kind of mobj can activate/deactivate the line.
ltf_monster_cross	a:0x40 d:0x80000	Cross events (mobj origin crosses the line while moving) from mobjs that have the MF_COUNTKILL flag can activate/deactivate the line.
ltf_player_cross	a:0x80 d:0x100000	Cross events (mobj origin crosses the line while moving) from player mobjs can activate/deactivate the line.
ltf_missile_cross	a:0x100 d:0x200000	Cross events (mobj origin crosses the line while moving) from mobjs that have the MF_MISSILE flag can activate/deactivate the line.
ltf_player_hit	a:0x200 d:0x400000	Hit events (line blocks mobj movement) from player mobjs can activate/deactivate the line.
ltf_other_hit	a:0x400 d:0x800000	Hit events (line blocks mobj movement) from non-player mobjs can activate/deactivate the line.
ltf_monster_hit	a:0x800 d:0x1000000	Hit events (line blocks mobj movement) from mobjs that have the MF_COUNTKILL flag can activate/ deactivate the line.
ltf_missile_hit	a:0x1000 d:0x2000000	Hit events (line blocks mobj movement) from mobjs that have the MF_MISSILE flag can activate/deactivate the line.
ltf_any_hit	a:0x2000 d:0x4000000	Hit events (line blocks mobj movement) from any kind of mobj can activate/deactivate the line.
ltf_ticker	a:0x8000000 d:0x10000000	Ticker events (can happen up to 35 times per second; controlled by Ticker start time, Ticker end time and Ticker tics) can activate/deactivate the line.

2.4.3.3 Activation requirements

ltf_mobj_gone	0x20000000	Activation or deactivation of the line is impossible unless all the mobjs with the type specified with Thing type (Ap9) are either removed from the map or have died (their health is equal to or less than zero).
ltf_no_other_use_secret	0x40000000	If the line is flagged Secret (standard Doom line flag) it can't be activated or deactivated with Use events from non-player mobjs.
ltf_activator_type	0x80000000	The line can only be activated or deactivated with events whose originator is a mobj of the type specified with Thing type (Ap9).

2.4.4 Flags2

Type: flags

There are several flags that are used to control various aspects about the line type, including activation method and requirements and the time to execute the line's function. The Flags2 property uses the flag definitions that begin with ltf2_.

2.4.4.1 Line function

ltf2_when_act	0x1	The function of the line is executed when the line changes state from inactive to active, i.e. it's activated.
ltf2_when_deact	0x2	The function of the line is executed when the line changes state from active to inactive, i.e. it's deactivated.
ltf2_when_last	0x10	The function of the line can only be executed when the line's counter is one (1). Otherwise the function won't be

		processed at all.
ltf2_while_act	0x4	The function of the line is executed repeatedly while the line is active. Controlled with Ticker start time, Ticker end time and Ticker tics.
ltf2_while_inact	0x8	The function of the line is executed repeatedly while the line is inactive. Controlled with TIcker start time, Ticker end time and Ticker tics.

2.4.4.2 Activation requirements

ltf2_key1	0x20	The activator must be a player who has key 1 (Doom: blue keycard, Heretic: yellow key). If the player doesn't have the key the message "You need a [key]." will be shown.
ltf2_key2	0x40	The activator must be a player who has key 2 (Doom: yellow keycard, Heretic: green key). If the player doesn't have the key the message "You need a [key]." will be shown.
ltf2_key3	0x80	The activator must be a player who has key 3 (Doom: red keycard, Heretic: blue key). If the player doesn't have the key the message "You need a [key]." will be shown.
ltf2_key4	0x100	The activator must be a player who has key 4 (Doom: blue skull key, Heretic: not used). If the player doesn't have the key the message "You need a [key]." will be shown.
ltf2_key5	0x200	The activator must be a player who has key 5 (Doom: yellow skull key, Heretic: not used). If the player doesn't have the key the message "You need a [key]." will be shown.
ltf2_key6	0x400	The activator must be a player who has key 6 (Doom: red skull key, Heretic: not used). If the player doesn't have the key the message "You need a [key]." will be shown.
ltf2_line_act	0x800	All the lines referenced with Line act lref and Line act lrefd must be active or the event-based activation and deactivation of the line will fail.
ltf2_line_inact	0x1000	All the lines referenced with Line inact lref and Line inact lrefd must be inactive or the event-based activation and deactivation of the line will fail.
ltf2_color	0x2000	The activator of the line must be of the color specified with Color (Ap8) or the activation and deactivation of the line will fail.
ltf2_health_above	0x4000	Health of the line's activator must be above the value specified with Health above (Ap0) or the activation and deactivation of the line will fail.
ltf2_health_below	0x8000	Health of the line's activator must be below the value specified with Health below (Ap1) or the activation and deactivation of the line will fail.
ltf2_power_above	0x10000	The line's activator must be a player whose armor level is above the value specified with Power above (Ap2) or the activation and deactivation of the line will fail.
ltf2_power_below	0x20000	The line's activator must be a player whose armor level is above the value specified with Power below (Ap3) or the activation and deactivation of the line will fail.
ltf2_singleplayer	0x40000	The line can be activated and deactivated in single-player games.

ltf2_cooperative	0x80000	The line can be activated and deactivated in co-operative multiplayer games.
ltf2_deathmatch	0x100000	The line can be activated and deactivated in deathmatch multiplayer games.
ltf2_any_mode	0x1C0000	ltf2_singleplayer, ltf2_cooperative and ltf2_deathmatch combined. The line can be activated and deactivated regardless of game mode.
ltf2_easy	0x200000	The line can be activated and deactivated in easy skill levels (1 and 2).
ltf2_med	0x400000	The line can be activated and deactivated in the medium skill level (3).
ltf2_hard	0x800000	The line can be activated and deactivated in hard skill levels (4 and 5).
ltf2_any_skill	0xE00000	ltf2_easy, ltf2_med and ltf2_hard combined. The line can be activated and deactivated regardless of the skill level.
ltf2_any	0xFC0000	ltf2_any_mode and ltf2_any_skill combined. The line can be activated and deactivated regardless of game mode or skill level.

2.4.4.3 Behavior

ltf2_multiple	0x1000000	When the line is activated or deactivated, copy the state of the line to all the lines with a matching tag number. This can be used for instance with doors, if you want that activating one side of the door marks the other side active as well (so the door can't be re-opened while it's already opening).
ltf2_2sided	0x2000000	The line can be activated and deactivated from both sides. If this flag is not used, only the events that deal with the front side of the line are processed.
ltf2_global_a_msg	0x4000000	The activation message (Act message) will be sent to all players in the game.
ltf2_global_d_msg	0x8000000	The deactivation message (Deact message) will be sent to all players in the game.
ltf2_global_msg	0xC000000	ltf2_global_a_msg and ltf2_global_d_msg combined. Both the activation and deactivation messages (Act message and Deact message) will be sent to all players in the game.

2.4.5 Flags3

Type: flags

Reserved for future extensions.

2.4.6 Class

Type: flags

Sets the class of the line. Each class performs a specific function (see Classes). Only one of the ltc_ flags is allowed. Usage of integer parameters Ip0, Ip1, Ip2…Ip19, floating point parameters Fp0, Fp1, Fp2…Fp19 and string parameters Sp0, Sp1…Sp4 varies depending on the value of this property.

2.4.7 Type

Type: flags

Activation type of the line. One of the following values:

```
lat_timed_off
lat_timed_on
lat_flip
lat_flip_timed_off
lat_flip_timed_on
```

See Activation and deactivation for a description of the types.

2.4.8 Count

Type: integer

Initial value for the line's counter, which is decremented every time the line successfully processes an event that is not a Chain event. Once the counter reaches zero all events will fail. A negative value for this property means the counter is disabled, and won't be checked during the line's event processing.

2.4.9 Time

Type: float

Time to stay active or inactive, in seconds. Used with `lat_timed_*` and `lat_flip_timed_*` activation types.

2.4.10 Act tag

Type: integer

Activation tag number of this line type. Several types can have the same activation tag. This number can be used in line references (with for instance `lref_act_tagged`). All signed 32-bit integer values are accepted.

2.4.11 Health above

Alternative: Ap0

Type: integer

Activation parameter 0. Activator health must be above this if the `ltf2_health_above` flag is set.

2.4.12 Health below

Alternative: Ap1

Type: integer

Activation parameter 1. Activator health must be below this if the `ltf2_health_below` flag is set.

2.4.13 Power above

Alternative: Ap2

Type: integer

Activation parameter 2. Activator power (armor) must be above this if the `ltf2_power_above` flag is set.

2.4.14 Power below

Alternative: Ap3
Type: integer

Activation parameter 3. Activator power (armor) must be below this if the `ltf2_power_below` flag is set.

2.4.15 Line act lref

Alternative: Ap4
Type: flags

Reference to the lines that must be active for the activation or deactivation of this line to succeed. This property sets the type of the line reference. (See References) The value must be one of the Flag definitions that begin with `lref_`. Only used with the `ltf2_line_act` flag.

2.4.16 Line act lrefd

Alternative: Ap5
Type: integer

Data component of the reference to the required active lines. (See References) Only used with the `ltf2_line_act` flag.

2.4.17 Line inact lref

Alternative: Ap6
Type: flags

Reference to the lines that must be inactive for the activation or deactivation of this line to succeed. This property sets the type of the line reference. (See References) The value must be one of the Flag definitions that begin with `lref_`. Only used with the `ltf2_line_inact` flag.

2.4.18 Line inact lrefd

Alternative: Ap7
Type: integer

Data component of the reference to the required inactive lines. (See References) Only used with the `ltf2_line_inact` flag.

2.4.19 Color

Alternative: Ap8
Type: integer

Activation parameter 8. Used with the `ltf2_color` flag. Specifies the required player color of the activator.

2.4.20 Thing type

Alternative: Ap9
Type: string

Activation parameter 9. Used with the `ltf2_mobj_gone` flag. Specifies the type of thing the flag refers to. ID of the thing type (for example `POSSESSED`). Case sensitive.

2.4.21 Ticker start time

Type: float

Number of seconds measured from the beginning of the level to start processing Ticker events for the line.

2.4.22 Ticker end time

Type: float

Number of seconds measured from the beginning of the level to stop processing Ticker events for the line. If this value is less than or equal to zero, the processing of Ticker events will continue indefinitely.

2.4.23 Ticker tics

Type: integer

Interval between consecutive Ticker events, in 35 Hz game tics. A value of zero means a Ticker event is sent on every tick, i.e. 35 times per second.

2.4.24 Event chain

Type: integer

Event chain for this type of lines (see Chains). ID number of an XG line type.

2.4.25 Act chain

Type: integer

Activation chain for this type of lines (see Chains). ID number of an XG line type.

2.4.26 Deact chain

Type: integer

Deactivation chain for this type of lines (see Chains). ID number of an XG line type.

2.4.27 Wall section

Type: flags

Accepted values:

```
lws_none
lws_mid
lws_upper
lws_lower
```

Defines which part of the line is affected by Act texture and Deact texture. Combining the values is not allowed, so Act texture and Deact texture can only be used to change one part of the wall. Note that any textures of the line that begin with *SW1* or *SW2* (switch textures) are automatically swapped with their counterparts when the line is activated or deactivated, regardless of the Wall section setting.

2.4.28 Act texture

Type: string

Name of the texture to set to the section of the line specified with Wall section when the line is activated. Note that any textures of the line that begin with *SW1* or *SW2* (switch textures) are automatically swapped with their counterparts when the line is activated or deactivated, regardless of the Act texture property.

2.4.29 Deact texture

Type: string

Name of the texture to set to the section of the line specified with Wall section when the line is deactivated. Note that any textures of the line that begin with *SW1* or *SW2* (switch textures) are automatically swapped with their counterparts when the line is activated or deactivated, regardless of the Deact texture property.

2.4.30 Act sound

Type: string

ID of the sound that will be played when the line is activated. The sound will appear to originate from the center of the sector where the line belongs to.

2.4.31 Deact sound

Type: string

ID of the sound that will be played when the line is deactivated. The sound will appear to originate from the center of the sector where the line belongs to.

2.4.32 Act message

Type: string

Message that will be sent to the activator of the line. If the `ltf2_global_a_msg` flag is set, the message will be sent to all players in the game. If the activator is a missile, its originator will receive the message.

2.4.33 Deact message

Type: string

Message that will be sent to the deactivator of the line. If the `ltf2_global_d_msg` flag is set, the message will be sent to all players in the game. If the activator is a missile, its originator will receive the message.

2.4.34 Texmove angle

Type: float

Direction to scroll the texture of the line, given in degrees: 0=right, 90=up, 180=left and 270=down. All floating-point values are accepted, though, even negative ones.

2.4.35 Texmove speed

Type: float

Speed of texture scrolling as pixels per game tic (35 Hz). The speed of 2 would scroll the texture 70 pixels per second.

2.4.36 Ip0 ... Ip19

Type: integer or string

Integer or string parameters (which will be translated to various integer values) for the line's function. Usage depends on the class of the line.

2.4.37 Fp0 ... Fp19

Type: float

Floating point parameters for the line's function. Usage depends on the class of the line.

2.4.38 Sp0 ... Sp4

Type: string

String parameters for the line's function. Usage depends on the class of the line.

2.5 Classes

The class of a line type defines its function. Each type belongs to a single class and thus performs only one function, like initiating a plane move or changing the type of a sector. The Class property of the DED Line Type definition selects the class to use. DED definitions of the available line classes are listed below.

```
Flag { ID = "ltc_none"; }
Flag { ID = "ltc_chain_sequence"; Value = 0x1; }
Flag { ID = "ltc_plane_move"; Value = 0x2; }
Flag { ID = "ltc_build_stairs"; Value = 0x3; }
Flag { ID = "ltc_damage"; Value = 0x4; }
Flag { ID = "ltc_power"; Value = 0x5; }
Flag { ID = "ltc_line_type"; Value = 0x6; }
Flag { ID = "ltc_sector_type"; Value = 0x7; }
Flag { ID = "ltc_sector_light"; Value = 0x8; }
Flag { ID = "ltc_activate"; Value = 0x9; }
Flag { ID = "ltc_key"; Value = 0xA; }
Flag { ID = "ltc_music"; Value = 0xB; }
Flag { ID = "ltc_sound"; Value = 0x14; }
Flag { ID = "ltc_line_count"; Value = 0xC; }
Flag { ID = "ltc_end_level"; Value = 0xD; }
Flag { ID = "ltc_disable_if_active"; Value = 0xE; }
Flag { ID = "ltc_enable_if_active"; Value = 0xF; }
Flag { ID = "ltc_explode"; Value = 0x10; }
Flag { ID = "ltc_plane_texture"; Value = 0x11; }
Flag { ID = "ltc_wall_texture"; Value = 0x12; }
Flag { ID = "ltc_command"; Value = 0x13; }
```

Note to people who will modify the source code and add new line classes: start using numbers from 0x10000. Classes 0 to 0xFFFF will be reserved for standardized use as specified in this document.

2.5.1 ltc_none

The line does nothing.

2.5.2 ltc_chain_sequence

A special class of line that sends Chain events to a copy of itself while active. This allows lines that perform a timed sequence of actions. If the line is deactivated the processing of the chain sequence is stopped. Each of the Chain events will be an activating event.

Ip0	Flags	chsf_done_d disables the line after the chain sequence has been completed. chsf_loop makes the sequence loop continuously.
Ip1…Ip19	Integers	List of chains. ID numbers of line types. A zero ends the list.
Fp0	Float	Randomness of intervals, in percents. A value of 100 means the actual interval of two chains is 0%…200% of the defined interval.
Fp1…Fp19	Floats	List of intervals, in seconds. They define the number of seconds to wait before processing the corresponding Chain event. For example, if Fp1 is

		2 and Ip1 is 5210, the chain 5210 is processed two seconds after the activation of the chain sequence.

2.5.3 ltc_plane_move

Moves a plane (floor or ceiling) of a sector to a given height. The texture of the plane can be changed in the beginning or in the end of the move. A sound can be played when the move begins or ends, or at a given interval while the plane is moving. The type of the sector whose plane is being moved can be changed in the beginning and in the end of the move.

Another way to move planes is the `ltc_build_stairs` class.

Ip0 Ip1	Lpref Lprefd	Reference to planes to move.
Ip2	Spref	Destination height (spref). `spref_none` means the destination height is zero. Offseted by Fp2.
Ip3	Flags	`pmf_crush` makes the plane crush things if they won't fit in the sector. `pmf_abort_a` will activate the line that initiated the plane move if the moving of the plane is aborted. Non-crushing plane moves are aborted if a thing doesn't fit in the sector (for instance, someone is standing under a door while it's closing). Crushers won't be aborted. `pmf_abort_d` will deactivate the line that initiated the plane move if the moving of the plane is aborted. `pmf_done_a` will activate the line that initiated the plane move when the plane reaches its destination height. `pmf_done_d` will deactivate the line that initiated the plane move when the plane reaches its destination height. `pmf_follow` makes the other plane of the sector follow the moving one, keeping the height of the sector the same throughout the move. `pmf_setorig` will update the plane's original height value after the move is done. `pmf_1snd` allows the playing of sounds only for the first referenced plane.
Ip4	Sound	Start sound. Name of the sound to play when the move is begun.
Ip5	Sound	End sound. Name of the sound to play when the move finishes.
Ip6	Sound	Move sound. Name of the sound to play while moving.
Ip7	Spref	Spref to the texture that is set when the move is begun.
Ip8	Flat	Name of the flat (texture) to set when the move is begun. This is used if Ip7 is set to `spref_special`.
Ip9	Spref	Spref to the texture that is set when the move finishes.
Ip10	Flat	Name of the flat (texture) to set when the move is begun. This is used if Ip9 is set to `spref_special`.
Ip11 Ip12	Lpref Lprefd	Reference to the sector whose type will be used for the moving plane's sector after the move is begun. `lpref_special` treats Ip12 as a sector type ID number.
Ip13 Ip14	Lpref Lprefd	Reference to the sector whose type will be used for the moving plane's sector after the move has finished. `lpref_special` treats Ip14 as a sector type ID number.
Fp0	Float	Move speed as units per (35 Hz) tic. A speed of 2 will move the plane 70 units per second.

Fp1	Float	Move speed while crushing, as units per tic.
Fp2	Float	Offset to destination height. Positive values move the destination upwards.
Fp3	Float	Move sound minimum interval, in seconds.
Fp4	Float	Move sound maximum interval, in seconds. The move sound will be played at random intervals, the minimum defined with Fp3 and the maximum with Fp4.
Fp5	Float	Time to wait before moving the plane, in seconds.
Fp6	Float	Wait time increment for each plane that gets moved. Has an effect only if more than one sector is being affected.

2.5.4 ltc_build_stairs

Forms incremental structures (stairs) from one or more series of planes. A sound can be played at the beginning of the stair building, before starting to move each step, while a step is moving and when a step reaches its destination.

Ip0 Ip1	Lpref Lprefd	Reference to planes where stair building will begin.
Ip2	Integer	If true (non-zero), stair building will only spread to planes with the same texture as the first plane.
Ip3	Integer	If true (non-zero), stair building spreads to all adjoining sectors that have a line facing the current sector (spread build). If false (zero), stair building only spreads over the line that is facing the current sector and has the lowest index number.
Ip4	Sound	Start build sound. Name of the sound to play when the building begins.
Ip5	Sound	Step start sound. Name of the sound to play when a step starts to move.
Ip6	Sound	Step end sound. Name of the sound to play when a step reaches its destination height.
Ip7	Sound	Step move sound. Name of the sound to play while a step is moving. Each step will play its own sound!
Fp0	Float	Build speed (units per tic). The first step will move to its destination height using this speed. A speed of 2 would move the step 70 units per second.
Fp1	Float	Step height. Both positive and negative values are accepted. This is the height difference between two consecutive steps.
Fp2	Float	Time to wait before starting to move the first step, in seconds.
Fp3	Float	Time to wait between steps, in seconds.
Fp4	Float	Move sound minimum interval, in seconds. The step move sound is played at random intervals, this (Fp4) setting the minimum and Fp5 setting the maximum.
Fp5	Float	Move sound maximum interval, in seconds.
Fp6	Float	Build speed delta per step. The step build speed will be modified with this value before starting to move each step (except the first one). Both positive and negative values are accepted.

2.5.5 ltc_damage

Damages or heals the activator.

Ip0	Integer	Minimum amount of damage to deal. Use negative values to heal the

		activator. The real amount of damage is a random number between the minimum and maximum.
Ip1	Integer	Maximum amount of damage to deal. Use negative values to heal the activator.
Ip2	Integer	Minimum required activator health. If it's below this, nothing will be done. Can be used to make the function non-lethal, for example.
Ip3	Integer	Maximum activator health. When healing, the activator's health won't rise above this.

2.5.6 ltc_power

Modifies activator's power (armor) level. Only operates on players.

Ip0	Integer	Minimum power delta. The real delta will be a random number between the minimum and maximum.
Ip1	Integer	Maximum power delta.
Ip2	Integer	Lower power limit. The player's power won't go below this.
Ip3	Integer	Upper power limit. The player's power won't go above this.

2.5.7 ltc_line_type

Changes the type of one or more lines.

Ip0 Ip1	Lref Lrefd	Reference to the lines whose type to change.
Ip2	Integer	Line type ID. Must be an XG line type.

2.5.8 ltc_sector_type

Changes the type of one or more sectors.

Ip0 Ip1	Lsref Lsrefd	Reference to the sectors whose type to change.
Ip2	Integer	Sector type ID. Must be an XG sector type, or zero.

2.5.9 ltc_sector_light

Changes the light level and color of one or more sectors.

Ip0 Ip1	Lsref Lsrefd	Reference to the sectors whose type to change.
Ip2	Integer	If non-zero, the light level of the sectors will be changed. Otherwise the light level won't be modified.
Ip3	Integer	If non-zero, the light color of the sectors will be changed. Otherwise the color of the light won't be modified.
Ip4	Lightref	Source of light level. To set the light level to an absolute value, set this to `lightref_none` and use Ip5 to specify the level.
Ip5	Integer	Offset to Ip4 (0…255).
Ip6	Lightref	Source of light color. Only `lightref_none`, `lightref_my` and `lightref_original` can be used.
Ip7	Integer	Offset to the red component of sector light color (0…255).

Ip8	Integer	Offset to the green component of sector light color (0…255).
Ip9	Integer	Offset to the blue component of sector light color (0…255).

2.5.10 ltc_activate

Sends a Chain event to the referenced lines. The current line's activator is used as the activator for each of them.

Ip0 Ip1	Lref Lrefd	Reference to one or more lines. Each one will receive a Chain event.

2.5.11 ltc_key

Gives keys to or takes them from the activator, who must be a player.

Ip0	Integer	Bitfield of the keys to give. Bit 1 (0x1) corresponds key 1, bit 2 (0x2) key 2, etc.
Ip1	Integer	Bitfield of the keys to take away.

2.5.12 ltc_music

Changes the music.

Ip0	Music	Name (ID) of the music to start playing.
Ip1	Integer	If non-zero, the music will play looped.

2.5.13 ltc_sound

Plays a sound in one or more sectors.

Ip0 Ip1	Lsref Lsrefd	Reference to one or more sectors.
Ip2	Sound	Name of the sound to play.

2.5.14 ltc_line_count

Modifies the counters of the referenced lines.

Ip0 Ip1	Lref Lrefd	Reference to one or more lines.
Ip2	Integer	If zero, the value in Ip3 is a delta. Otherwise the counters of the referenced lines are set to the exact value in Ip4.
Ip3	Integer	Delta or the new counter value. If Ip2 is zero, this is treated as a signed delta that is added to the current counter value of a line.

2.5.15 ltc_end_level

Exits the current level.

Ip0	Integer	If non-zero, the next level will be the secret level (Secret Exit).

2.5.16 ltc_disable_if_active

If the line is active, disables all the referenced lines. If it's inactive, the referenced lines are enabled.

Ip0 Ip1	Lref Lrefd	Reference to one or more lines.

2.5.17 ltc_enable_if_active

If the line is active, enables all the referenced lines. If it's inactive, the referenced lines are disabled.

Ip0 Ip1	Lref Lrefd	Reference to one or more lines.

2.5.18 ltc_explode

Kills the activator. This is done by calling the P_ExplodeMissile() routine, which basically kills a thing by changing its state to the Death state of the thing type.

2.5.19 ltc_plane_texture

Changes the texture of the referenced planes.

Ip0 Ip1	Lpref Lprefd	Reference to one or more planes.
Ip2	Spref	Spref to the new texture. Only sprefs that refer to planes can be used.
Ip3	Flat	Name of the flat (texture) that will be used if Ip2 is `spref_none`.

2.5.20 ltc_wall_texture

Changes the textures of the referenced lines.

Ip0 Ip1	Lref Lrefd	Reference to one or more lines.
Ip2	Integer	Zero or one. Specifies the line side to modify. Side zero is the front side.
Ip3	Texture	Name of the new upper texture. If not specified, the upper textures of the lines won't be affected.
Ip4	Texture	Name of the new middle texture. If not specified, the middle textures of the lines won't be affected.
Ip5	Texture	Name of the new lower texture. If not specified, the lower textures of the lines won't be affected.

2.5.21 ltc_command

Executes a console command.

Sp0	String	The console command to be executed. Note that semicolons can be used to separate any number of commands, so this is perfectly legal: `"fog on; fog end 500"`

3 Sector types

An XG sector is a normal Doom sector whose type is an XG sector type. XG sectors have special physics processing for wind, customized gravity and friction. Each sector has a number of chains that are used to define extended functionality. Note that sector chains refer to line types, so anything that can be achieved with line types can also be done with sectors. For instance, if you want a sector to damage all the things touching its floor, you would set the sector type's floor chain to a line type that deals damage.

3.1 Physics

3.1.1 Wind

Wind increases the momentum of things inside a sector. Wind speed is defined as the increase in momentum per game tic. There are two kinds of wind: horizontal and vertical. Horizontal wind pushes things on the XY plane to a given direction specified with an angle. Vertical wind affects Z momentum. Standard Doom gravity corresponds a vertical wind of –1. Wind can be configured to only affect certain types of map objects.

If you want to synchronize wind speed with a scrolling texture (for instance to create the illusion that a scrolling floor is moving the player around), use the following equation (f is the sector friction).

$$W_{spd} = T_{spd}(1-f)$$

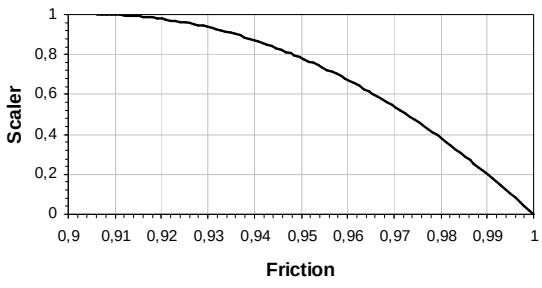
3.1.2 Gravity

Gravity pulls things downwards. A sector type can define a custom gravity, which is used for all things inside the sector. Standard gravity is one (1).

3.1.3 Friction

The friction of a sector determines how easy it is to change the momentum of things that touch the floor of the sector. Friction is defined as a floating point number in range 0 to 1. The lower the number, the larger the friction seems. Each game tic (35 times per second) the momentum of the mobjs that touch the floor of the sector is multiplied by the friction value. This means if the friction is set to one, the momentum of the things inside the sector won't decrease at all.

Standard friction	0xE800	0.90625
Heretic ice friction	0xF900	0.97265625



As friction decreases (the value is nearing one), it becomes harder and harder to change the momentum of things, i.e. to

accelerate or decelerate. The graph shows how momentum changes are scaled.

3.2 Chains

Each XG sector type has four chains: the floor, ceiling, inside and ticker chain. Like with lines, chains are always line type IDs. The floor chain is processed when a map object of a given type is touching the floor: a chain event whose originator is the object in question is sent to a temporary line that appears to belong to the sector. The ceiling chain works similarly, but is processed when an object is touching the ceiling. The inside chain is processed for all objects inside the sector. The ticker chain is processed up to 35 times per second. For the ticker chain there is no event originator.

Each chain can be configured to only affect certain types of things (player, monster, missile, etc.). Also, each chain has a start time and an end time, which define when the chain is operating. The rate at which to send the Chain events can be set as a min/max interval pair. Each chain has a counter that can be used to limit the number of times the chain can be processed. If the counter is greater than zero, it will be decremented every time a Chain event is successfully processed.

The chains can be configured to send activating or deactivating events. Internally, this means the temporary line that receives the events is initially inactive or active, respectively. The temporary line is set up as follows: it is a one-sided line, whose front sector is the sector the chain belongs to. It has no sidedefs, so operations that modify the sides of the line (Act texture, for instance) have no effect. The line's type is equal to the chain being processed (for example 5210). The line's tag number is equal to the sector's tag number.

As an example, a sector type that deals one point of damage to all players inside the sector once per second would be set up in the following way. The floor chain would be a line type that, when activated, deals a point of damage to its activator. Flags for the floor chain would be `scef_player_a`, which causes an activating Chain event to be sent for player mobjs. The floor chain counter would be set to –1 to allow the chain to work indefinitely. Both the min and max intervals for the chain would be one.

3.3 Functions

Sector functions (XG functions) are functions in a mathematical sense. They evaluate to a number that might change as time goes by. XG functions are strings. Some characters are interpreted as control characters and others represent values from zero to one. XG sectors can use functions to change their light level, color, the height of the floor and ceiling planes and to send Chain events.

3.3.1 Symbols

The table below lists all the symbols that can be used in functions. There can be any number of spaces between two symbols. Often adding a couple of spaces makes the function easier for humans to understand.

a...z	Values from zero to one. Values between two consecutive lowercase letters will be interpolated. Example: az will smoothly blend from zero to one, but aZ and Az will not.
A...Z	Values from zero to one. Won't be interpolated between steps.
/	Exact value. A floating point number must immediately follow. Corresponds a lowercase letter, so interpolation will be done. Example: `z/0.355a` would interpolate from 1 to 0.355 and then to zero. `/1` is the same as z.

%	Exact value. A floating point number must immediately follow. Corresponds an uppercase letter, so there will be no interpolation. Example: %1 is the same as Z.
(number)	Repeat the following value this many times. The value must follow the number immediately. Example: 12A equals AAAAAAAAAAAA. The smallest number that can be used this way is one, but 1a equals a, so there isn't much point in using it.
.	Break interpolation. Placing a period after a lowercase letter will cause the evaluator to skip the letter. Example: azaz will smoothly blend from 0 to 1, then from 1 to 0 and finally from 0 to 1. az.az will smoothly blend from 0 to 1 and then again from 0 to 1. The second interpolation that would go from 1 to 0 is broken by the period.
#	Set step timer. A number must immediately follow a # symbol. Example: AB#14CD will set the step timer for the C step to 14 tics. This means the step will be in effect for 14 tics (0.4 seconds).
?	Random step timer. A number must immediately follow a ? symbol. The step timer for the next step of the function will be set to a random number between zero and the given value. Example: AB?6CD will set the step timer for the C step to a random number between zero and six.
>	Begin of repeat sequence marker. The evaluator skips this.
<	Repeat. Makes the evaluator jump to the beginning of the function or to the latest > marker. Example: az>mz< will smoothly blend from 0 to 1, then from 1 to 0.48 and finally repeat blending back and forth between 0.48 and 1. AZ< as a light function could be used for a strobo effect.
!	Chain event. A line type ID must immediately follow the exclamation point !. The event will be an activating one and it will be sent in the beginning of the step (when the evaluator arrives at the value following the ! symbol). Example: a!5210z will first interpolate from 0 to 1 and then send the Chain event 5210.

The first character of a function string can also be a special character:

=	Linked function. The value of the function is linked to the value of another function of the same sector (their values are always equal). The character following the equal sign = determines which function this one will be linked to: f Floor fn c Ceiling fn r Red fn g Green fn b Blue fn l Light fn Example: the function =b would be linked to the Blue function of the sector.
+	Dynamic offset. When the function is initialized (at the beginning of the level or when the sector type is set) its offset will be modified depending on the character following the plus sign +. Evaluation of the function will begin from the third character of the function string. f Original floor height c Original ceiling height r Original red color g Original green color b Original blue color l Original light level Example: beginning a function with +f would add the original floor height of the sector to the function offset.

3.3.2 Evaluation of functions

The first actual symbol of a (non-linked) function string must be a value (an uppercase or a lowercase letter or an exact value). It is the initial value of the function.

Each value symbol defines a step. The evaluator advances to the next step when the timer for the current one reaches zero. The type definition specifies the minimum and maximum number of tics for each step. If the step timer isn't set with the # or ? symbols, it will be set to a random number between the minimum and maximum tics for each step. Interpolation is always done between the current step and the next one. If both are interpolateable (lowercase letter or an exact value defined with a slash /) the value of the function will gradually blend from the current step's value to the next one's.

The current value of a step will be multiplied with a scaling factor and an offset will be added. Acceptable values for the function depend on the context: for example, light and color functions are automatically scaled with 255 and have no offset. The floor and ceiling functions' scale and offset can be specified in the type definition.

Examples:

zaN	Interpolate from 1 to 0, wait one step at 0 (won't interpolate from a to N) and stay at 0.52.
za.N	Interpolate from 1 to 0, stay at 0.52.
A.Z	The same as AZ: one step at 0, stay at 1.
az.<	Repeat interpolating from 0 to 1. The period breaks the interpolation that would happen from 1 to 0 (over the repeat marker).
+f m ?35m > miecbceimqtwyzywtq <	Dynamic offset to the original floor height of the sector. Initial value (the first step) is 0.48. The second step has the same value, but a random timer with at most 35 tics (one second). Start repeating an interpolated approximation of a sine wave.
z5/1>#1%0.333?7%1<	Initial value is 1. The first step is followed by 5 steps, each of which has the exact value of 1 (same as z). Start repeating a sequence whose first step is one tic long and has the exact non-interpolating value of 0.333. The second step of the sequence is a random length one (up to 7 tics) with the non-interpolating exact value of one (same as Z).

3.4 The Sector Type definition

3.4.1 ID

Type: integer

A unique integer number identifying this sector type. The value is used in level editors to refer to this type. Note that XG sector types take precedence, so if the ID number is the same as that of a normal Doom sector type, the XG version will be used. IDs must be in range from 1 to 65535.

3.4.2 Comment

Type: string

DED Manager will show this comment in the lookup dropdown lists and in the information section of the main index, but Doomsday itself does not use this string for

anything. It's a good idea to make the Comment a short description of what the sector type does.

3.4.3 Flags

Type: flags

Uses the flags that begin with `stf_`.

3.4.3.1 General

`stf_gravity`	0x1	The custom gravity set with the Gravity property is used inside the sector. Otherwise the map's general gravity is in effect.
`stf_friction`	0x2	The custom friction set with the Friction property is used inside the sector. Otherwise the standard friction of 0.90625 (fixed point 0xE800) is used.
`stf_crush`	0x4	The sector will crush things that don't fit inside it after moving the floor or ceiling plane with Floor fn / Ceiling fn.
`stf_tagtexmove`	0x80	Texture movement angle for both the ceiling and floor planes is determined by finding the first line that belongs to the sector and has the line tag number specified with Act tag. Textures will scroll in the direction of the line.

3.4.3.2 Wind

`stf_windplayer`	0x8	Wind affects all player mobjs inside the sector.
`stf_windother`	0x10	Wind affects all non-player mobjs inside the sector.
`stf_windmonster`	0x20	Wind affects all mobjs with the `MF_COUNTKILL` flag inside the sector.
`stf_windmissile`	0x40	Wind affects all mobjs with the `MF_MISSILE` flag inside the sector.
`stf_windany`	0x18	Wind affects all mobjs inside the sector.
`stf_tagwind`	0x100	Wind direction is determined by finding the first line that belongs to the sector and has the line tag number specified with Act tag. Wind will blow in the direction of the line.
`stf_floorwind`	0x200	Wind only affects mobjs that are touching the floor.
`stf_ceilingwind`	0x400	Wind only affects mobjs that are touching the ceiling.

3.4.4 Act tag

Type: integer

Activation tag number for the sector type. Several types can have the same activation tag. This number can be used in sector references (with for instance `lsref_act_tagged`). All signed 32-bit integer values are accepted.

3.4.5 Floor/Ceiling/Inside/Ticker chain…

The four chains of the sector are all defined in the same way. The "X" in the headings below can be replaced with Floor, Ceiling, Inside or Ticker.

3.4.5.1 X chain

Type: integer

ID number of an XG line type (see Chains).

3.4.5.2 X chain flags

Type: flags

Sector chain flags are prefixed `scef_` (means Sector Chain Event Flag). There is a `_a` and `_d` version of each flag. Both versions of the same flag can't be used.

`scef_player`	a:0x1 d:0x40	Player mobjs cause a Chain event to be sent. `scef_player_a` will send an activating event and `scef_player_d` a deactivating event.
`scef_other`	a:0x2 d:0x80	Non-player mobjs cause a Chain event to be sent.
`scef_monster`	a:0x4 d:0x100	Mobjs that have the `MF_COUNTKILL` flag cause a Chain event to be sent.
`scef_missile`	a:0x8 d:0x200	Mobjs that have the `MF_MISSILE` flag cause a Chain event to be sent.
`scef_any`	a:0x10 d:0x400	Any mobj cause a Chain event to be sent.
`scef_ticker`	a:0x20 d:0x800	Use only with the Ticker chain. Chain events will be sent based on the game ticker, up to 35 times per second. The chain start time, chain end time and chain min/max interval properties can be used to configure how often the ticker sends the Chain events.

3.4.5.3 X chain start time

Type: float

Number of seconds measured from the beginning of the level to the moment when the chain begins operating.

3.4.5.4 X chain end time

Type: float

Number of seconds measured from the beginning of the level to the moment when the chain stops operating. If this is equal to or less than zero, the chain will continue operating indefinitely.

3.4.5.5 X chain min interval

Type: float

Minimum interval, in seconds, between two consecutive Chain events. The real interval is a random number between the minimum and maximum intervals.

3.4.5.6 X chain max interval

Type: float

Maximum interval, in seconds, between two consecutive Chain events.

3.4.5.7 X chain count

Type: integer

Number of times the chain can be processed successfully. When the chain counter becomes zero, the chain will stop operating. Negative values mean the chain counter is disabled and it will continue to work until the chain end time passes.

3.4.6 Ambient sound

Type: string

ID of the ambient sound of the sector. The sound will be played at given intervals and appears to originate from the center of the sector.

3.4.7 Ambient min interval

Type: float

Minimum interval in playing the ambient sound of the sector, in seconds. The real interval is a random number between the minimum and maximum intervals.

3.4.8 Ambient max interval

Type: float

Maximum interval in playing the ambient sound of the sector, in seconds.

3.4.9 Floor texmove angle

Type: float

Direction for floor texture scrolling, in degrees: 0=east, 90=north, 180=west and 270=south. All floating-point values are accepted, though, even negative ones. This is not used if the `stf_tagtexmove` flag is set.

3.4.10 Floor texmove speed

Type: float

Speed of floor texture scrolling as pixels per game tic (35 Hz). The speed of 2 would scroll the texture 70 pixels per second.

3.4.11 Ceiling texmove angle

Type: float

Direction for ceiling texture scrolling, in degrees: 0=east, 90=north, 180=west and 270=south. All floating-point values are accepted, though, even negative ones. This is not used if the `stf_tagtexmove` flag is set.

3.4.12 Ceiling texmove speed

Type: float

Speed of ceiling texture scrolling as pixels per game tic (35 Hz). The speed of 2 would scroll the texture 70 pixels per second.

3.4.13 Wind angle

Type: float

Direction of wind inside the sector, in degrees: 0=east, 90=north, 180=west and 270=south. All floating point values are accepted, though, even negative ones. This is not used if the `stf_tagwind` flag is set.

3.4.14 Wind speed

Type: float

Horizontal wind speed. The unit for wind speed is units/tic^2 (see Wind).

3.4.15 Vertical wind

Type: float

Vertical wind speed. Positive speed lifts things upward, negative one pushes them down. The unit for wind speed is units/tic^2 (see Wind).

3.4.16 Gravity

Type: float

Gravity inside the sector. Only used if the `stf_gravity` flag is set.

3.4.17 Friction

Type: float

Friction inside the sector. Only used if the `stf_friction` flag is set.

3.4.18 Light/Red/Green/Blue fn…

These functions are used to control the light level and color of the sector (see Functions). The "X" in the headings below can be replaced with Light, Red, Green or Blue.

3.4.18.1 X fn

Type: string

The function string.

3.4.18.2 X fn min/max tics

Type: integer

Minimum/maximum number of tics between evaluation steps. The real number of tics is a random number between the minimum and maximum tics.

3.4.19 Floor/Ceiling fn…

These functions are used to control the floor and ceiling planes of the sector (see Functions). The "X" in the headings below can be replaced with either Floor or Ceiling. The value of the function directly becomes the new height of the plane in question.

3.4.19.1 X fn

Type: string

The function string.

3.4.19.2 X fn scale

Type: float

The value of the function (usually between 0 and 1) will be multiplied with this factor.

3.4.19.3 X fn offset

Type: float

This number will be added to the value of the function. This is a fixed offset unlike the dynamic offset that can be set with the function string (for example `+f`).

3.4.19.4 X fn min/max tics

Type: integer

Minimum/maximum number of tics between evaluation steps. The real number of tics is a random number between the minimum and maximum tics.

4 Examples

4.1 Line types

```
Line Type {
  ID = 6101;
  Flags2 = "ltf2_when_deact ltf2_when_act ltf2_any";
  Class = "ltc_sector_light";
  Type = "lat_flip";
  Count = -1;
  Ip0 = "lpref_line_tagged_floors";
  Ip2 = 1;
  Ip3 = 1;
  Ip4 = "lightref_original";
  Ip6 = "lightref_original";
}
```

Restores the light level and color of the sectors that have the same tag number as the line to their original values, when the line is activated or deactivated. The line can only be activated with Chain events (no activation method is specified). Game mode and skill level have no effect. The line can be activated an infinite number of times (count is –1). Note that a line-plane reference is used to refer to whole sectors. This can be done because `lpref_line_tagged_floors` is actually the same as `lsref_line_tagged`.

```
Line Type {
  ID = 5070;
  Flags = "ltf_player_use";
  Flags2 = "ltf2_when_act ltf2_any";
  Class = "ltc_wall_texture";
  Type = "lat_timed_off";
  Count = 1;
  Time = 1;
  Act sound = "swtchn";
  Ip0 = "lref_all";
  Ip3 = "SP_FACE1";
  Ip4 = "SP_FACE1";
  Ip5 = "SP_FACE1";
}
```

The line can be activated with player Use events. Game mode and skill level have no effect. The line is initially inactive and once activated it stays active for one second after which it deactivates itself. When activated, the upper, lower and middle textures of the front side of all lines in the map are changed to *SP_FACE1*. The standard Doom switch sound *swtchn* is played upon activation. The line can only be activated once.

```
Line Type {
  ID = 5006;
  Comment = "Lower when Poss gone";
  Flags = "ltf_mobj_gone ltf_ticker";
  Flags2 = "ltf2_when_act ltf2_any";
  Class = "ltc_plane_move";
  Type = "lat_timed_off";
  Count = 1;
```

```
  Time = 1;
  Thing type = "POSSESSED";
  Ticker tics = 5;
  Texmove angle = 90;
  Texmove speed = 1;
  Ip0 = "lpref_my_floor";
  Ip2 = "spref_lowest_floor";
  Ip3 = "pmf_follow pmf_crush";
  Ip4 = "bdopn";
  Ip5 = "bdcls";
  Ip6 = "punch";
  Ip10 = "HCSKULL1";
  Fp0 = 3;
  Fp1 = 0.2;
  Fp3 = 0.2;
  Fp4 = 0.4;
}
```

The line is activated by the game ticker. Activation will only succeed if all the things whose type is `POSSESSED` have died or been removed from the map. Game mode and skill level have no effect. When activated, the line begins moving the floor plane of its sector. The destination height is the lowest floor height of the adjacent sectors. The ceiling of the sector will follow at a fixed distance from the floor, so the height of the sector won't change. The sector would crush things if they got in the way. The sound *bdopn* is played when the move is begun, and *bdcls* is played when the destination height has been reached. The *punch* sound is played at random intervals while the plane is moving (min: 0.2 seconds, max: 0.4 seconds). The planes will be moved 3 units per game tic, which means they move 105 units per second. In the end of the move the texture *HCSKULL1* is applied to the floor.

4.2 Sector types

```
Sector Type {
  ID = 5001;
  Comment = "Scroll along tag 9999";
  Flags = "stf_tagtexmove stf_tagwind stf_windmissile stf_windplayer";
  Act tag = 9999;
  Floor texmove speed = 1.8;
  Wind speed = 0.16875;
}
```

The sector has a scrolling floor and wind that affects player mobjs and missiles. The wind is blowing in the same direction as the floor scrolls, which is the direction the sector's line that has the tag 9999 is pointing. Wind speed has been synchronized with the scrolling speed, so it appears that the floor is moving and moves the player with it.

```
Sector Type {
  ID = 5002;
  Ticker chain = 6101;
  Ticker chain count = 1;
}
```

The sector will immediately send the Chain event 6101. It will only be sent once. If the definition of 6101 that was listed in the line type examples is used, this sector type could be used to restore the original light level and color of a sector.

5 Definitions

5.1 Line types

```
Flag { ID = "ltc_none"; }
Flag { ID = "ltc_chain_sequence"; Value = 0x1; }
Flag { ID = "ltc_plane_move"; Value = 0x2; }
Flag { ID = "ltc_build_stairs"; Value = 0x3; }
Flag { ID = "ltc_damage"; Value = 0x4; }
Flag { ID = "ltc_power"; Value = 0x5; }
Flag { ID = "ltc_line_type"; Value = 0x6; }
Flag { ID = "ltc_sector_type"; Value = 0x7; }
Flag { ID = "ltc_sector_light"; Value = 0x8; }
Flag { ID = "ltc_activate"; Value = 0x9; }
Flag { ID = "ltc_key"; Value = 0xA; }
Flag { ID = "ltc_music"; Value = 0xB; }
Flag { ID = "ltc_sound"; Value = 0x14; }
Flag { ID = "ltc_line_count"; Value = 0xC; }
Flag { ID = "ltc_end_level"; Value = 0xD; }
Flag { ID = "ltc_disable_if_active"; Value = 0xE; }
Flag { ID = "ltc_enable_if_active"; Value = 0xF; }
Flag { ID = "ltc_explode"; Value = 0x10; }
Flag { ID = "ltc_plane_texture"; Value = 0x11; }
Flag { ID = "ltc_wall_texture"; Value = 0x12; }
Flag { ID = "ltc_command"; Value = 0x13; }
Flag { ID = "ltf_active"; Value = 0x1; }
Flag { ID = "ltf_player_use_a"; Value = 0x2; }
Flag { ID = "ltf_other_use_a"; Value = 0x4; }
Flag { ID = "ltf_player_shoot_a"; Value = 0x8; }
Flag { ID = "ltf_other_shoot_a"; Value = 0x10; }
Flag { ID = "ltf_any_cross_a"; Value = 0x20; }
Flag { ID = "ltf_monster_cross_a"; Value = 0x40; }
Flag { ID = "ltf_player_cross_a"; Value = 0x80; }
Flag { ID = "ltf_missile_cross_a"; Value = 0x100; }
Flag { ID = "ltf_player_hit_a"; Value = 0x200; }
Flag { ID = "ltf_other_hit_a"; Value = 0x400; }
Flag { ID = "ltf_monster_hit_a"; Value = 0x800; }
Flag { ID = "ltf_missile_hit_a"; Value = 0x1000; }
Flag { ID = "ltf_any_hit_a"; Value = 0x2000; }
Flag { ID = "ltf_player_use_d"; Value = 0x4000; }
Flag { ID = "ltf_other_use_d"; Value = 0x8000; }
Flag { ID = "ltf_player_shoot_d"; Value = 0x10000; }
Flag { ID = "ltf_other_shoot_d"; Value = 0x20000; }
Flag { ID = "ltf_any_cross_d"; Value = 0x40000; }
Flag { ID = "ltf_monster_cross_d"; Value = 0x80000; }
Flag { ID = "ltf_player_cross_d"; Value = 0x100000; }
Flag { ID = "ltf_missile_cross_d"; Value = 0x200000; }
Flag { ID = "ltf_player_hit_d"; Value = 0x400000; }
Flag { ID = "ltf_other_hit_d"; Value = 0x800000; }
Flag { ID = "ltf_monster_hit_d"; Value = 0x1000000; }
Flag { ID = "ltf_missile_hit_d"; Value = 0x2000000; }
Flag { ID = "ltf_any_hit_d"; Value = 0x4000000; }
Flag { ID = "ltf_player_use"; Value = 0x4002; }
```

```
Flag { ID = "ltf_other_use"; Value = 0x8004; }
Flag { ID = "ltf_player_shoot"; Value = 0x10008; }
Flag { ID = "ltf_other_shoot"; Value = 0x20010; }
Flag { ID = "ltf_any_cross"; Value = 0x40020; }
Flag { ID = "ltf_monster_cross"; Value = 0x80040; }
Flag { ID = "ltf_player_cross"; Value = 0x100080; }
Flag { ID = "ltf_missile_cross"; Value = 0x200100; }
Flag { ID = "ltf_player_hit"; Value = 0x400200; }
Flag { ID = "ltf_other_hit"; Value = 0x800400; }
Flag { ID = "ltf_monster_hit"; Value = 0x1000800; }
Flag { ID = "ltf_missile_hit"; Value = 0x2001000; }
Flag { ID = "ltf_any_hit"; Value = 0x4002000; }
Flag { ID = "ltf_ticker_a"; Value = 0x8000000; }
Flag { ID = "ltf_ticker_d"; Value = 0x10000000; }
Flag { ID = "ltf_ticker"; Value = 0x18000000; }
Flag { ID = "ltf_mobj_gone"; Value = 0x20000000; }
Flag { ID = "ltf_no_other_use_secret"; Value = 0x40000000; }
Flag { ID = "ltf_activator_type"; Value = 0x80000000; }
Flag { ID = "ltf2_when_act"; Value = 0x1; }
Flag { ID = "ltf2_when_deact"; Value = 0x2; }
Flag { ID = "ltf2_when_last"; Value = 0x10; }
Flag { ID = "ltf2_while_act"; Value = 0x4; }
Flag { ID = "ltf2_while_inact"; Value = 0x8; }
Flag { ID = "ltf2_key1"; Value = 0x20; }
Flag { ID = "ltf2_key2"; Value = 0x40; }
Flag { ID = "ltf2_key3"; Value = 0x80; }
Flag { ID = "ltf2_key4"; Value = 0x100; }
Flag { ID = "ltf2_key5"; Value = 0x200; }
Flag { ID = "ltf2_key6"; Value = 0x400; }
Flag { ID = "ltf2_line_act"; Value = 0x800; }
Flag { ID = "ltf2_line_inact"; Value = 0x1000; }
Flag { ID = "ltf2_color"; Value = 0x2000; }
Flag { ID = "ltf2_health_above"; Value = 0x4000; }
Flag { ID = "ltf2_health_below"; Value = 0x8000; }
Flag { ID = "ltf2_power_above"; Value = 0x10000; }
Flag { ID = "ltf2_power_below"; Value = 0x20000; }
Flag { ID = "ltf2_singleplayer"; Value = 0x40000; }
Flag { ID = "ltf2_cooperative"; Value = 0x80000; }
Flag { ID = "ltf2_deathmatch"; Value = 0x100000; }
Flag { ID = "ltf2_any_mode"; Value = 0x1C0000; }
Flag { ID = "ltf2_easy"; Value = 0x200000; }
Flag { ID = "ltf2_med"; Value = 0x400000; }
Flag { ID = "ltf2_hard"; Value = 0x800000; }
Flag { ID = "ltf2_any_skill"; Value = 0xE00000; }
Flag { ID = "ltf2_any"; Value = 0xFC0000; }
Flag { ID = "ltf2_multiple"; Value = 0x1000000; }
Flag { ID = "ltf2_2sided"; Value = 0x2000000; }
Flag { ID = "ltf2_global_a_msg"; Value = 0x4000000; }
Flag { ID = "ltf2_global_d_msg"; Value = 0x8000000; }
Flag { ID = "ltf2_global_msg"; Value = 0xC000000; }
Flag { ID = "lat_timed_off"; }
Flag { ID = "lat_timed_on"; Value = 0x1; }
Flag { ID = "lat_flip"; Value = 0x2; }
Flag { ID = "lat_flip_timed_off"; Value = 0x3; }
Flag { ID = "lat_flip_timed_on"; Value = 0x4; }
Flag { ID = "lws_none"; }
Flag { ID = "lws_mid"; Value = 0x1; }
Flag { ID = "lws_upper"; Value = 0x2; }
Flag { ID = "lws_lower"; Value = 0x3; }
Flag { ID = "lref_self"; }
Flag { ID = "lref_tagged"; Value = 0x1; }
Flag { ID = "lref_line_tagged"; Value = 0x2; }
Flag { ID = "lref_act_tagged"; Value = 0x3; }
```

```
Flag { ID = "lref_index"; Value = 0x4; }
Flag { ID = "lref_all"; Value = 0x5; }
Flag { ID = "lpref_none"; }
Flag { ID = "lpref_my_floor"; Value = 0x1; }
Flag { ID = "lpref_tagged_floors"; Value = 0x2; }
Flag { ID = "lpref_line_tagged_floors"; Value = 0x3; }
Flag { ID = "lpref_act_tagged_floors"; Value = 0x4; }
Flag { ID = "lpref_index_floor"; Value = 0x5; }
Flag { ID = "lpref_all_floors"; Value = 0x6; }
Flag { ID = "lpref_my_ceiling"; Value = 0x7; }
Flag { ID = "lpref_tagged_ceilings"; Value = 0x8; }
Flag { ID = "lpref_line_tagged_ceilings"; Value = 0x9; }
Flag { ID = "lpref_act_tagged_ceilings"; Value = 0xA; }
Flag { ID = "lpref_index_ceiling"; Value = 0xB; }
Flag { ID = "lpref_all_ceilings"; Value = 0xC; }
Flag { ID = "lpref_special"; Value = 0xD; }
Flag { ID = "lsref_my"; Value = 0x1; }
Flag { ID = "lsref_tagged"; Value = 0x2; }
Flag { ID = "lsref_line_tagged"; Value = 0x3; }
Flag { ID = "lsref_act_tagged"; Value = 0x4; }
Flag { ID = "lsref_index"; Value = 0x5; }
Flag { ID = "lsref_all"; Value = 0x6; }
Flag { ID = "spref_none"; }
Flag { ID = "spref_my_floor"; Value = 0x1; }
Flag { ID = "spref_my_ceiling"; Value = 0x2; }
Flag { ID = "spref_original_floor"; Value = 0x3; }
Flag { ID = "spref_original_ceiling"; Value = 0x4; }
Flag { ID = "spref_current_floor"; Value = 0x5; }
Flag { ID = "spref_current_ceiling"; Value = 0x6; }
Flag { ID = "spref_highest_floor"; Value = 0x7; }
Flag { ID = "spref_highest_ceiling"; Value = 0x8; }
Flag { ID = "spref_lowest_floor"; Value = 0x9; }
Flag { ID = "spref_lowest_ceiling"; Value = 0xA; }
Flag { ID = "spref_next_highest_floor"; Value = 0xB; }
Flag { ID = "spref_next_highest_ceiling"; Value = 0xC; }
Flag { ID = "spref_next_lowest_floor"; Value = 0xD; }
Flag { ID = "spref_next_lowest_ceiling"; Value = 0xE; }
Flag { ID = "spref_min_bottom_texture"; Value = 0xF; }
Flag { ID = "spref_min_mid_texture"; Value = 0x10; }
Flag { ID = "spref_min_top_texture"; Value = 0x11; }
Flag { ID = "spref_max_bottom_texture"; Value = 0x12; }
Flag { ID = "spref_max_mid_texture"; Value = 0x13; }
Flag { ID = "spref_max_top_texture"; Value = 0x14; }
Flag { ID = "lightref_none"; }
Flag { ID = "lightref_my"; Value = 0x1; }
Flag { ID = "lightref_original"; Value = 0x2; }
Flag { ID = "lightref_current"; Value = 0x3; }
Flag { ID = "lightref_highest"; Value = 0x4; }
Flag { ID = "lightref_lowest"; Value = 0x5; }
Flag { ID = "lightref_next_highest"; Value = 0x6; }
Flag { ID = "lightref_next_lowest"; Value = 0x7; }
Flag { ID = "chsf_done_d"; Value = 0x1; }
Flag { ID = "chsf_loop"; Value = 0x2; }
Flag { ID = "pmf_crush"; Value = 0x1; }
Flag { ID = "pmf_abort_a"; Value = 0x2; }
Flag { ID = "pmf_abort_d"; Value = 0x4; }
Flag { ID = "pmf_done_a"; Value = 0x8; }
Flag { ID = "pmf_done_d"; Value = 0x10; }
Flag { ID = "pmf_follow"; Value = 0x20; }
Flag { ID = "pmf_setorig"; Value = 0x40; }
Flag { ID = "pmf_1snd"; Value = 0x100; }
```

5.2 Sector types

```
Flag { ID = “stf_gravity”; Value = 0x1; }
Flag { ID = “stf_friction”; Value = 0x2; }
Flag { ID = “stf_crush”; Value = 0x4; }
Flag { ID = “stf_windplayer”; Value = 0x8; }
Flag { ID = “stf_windother”; Value = 0x10; }
Flag { ID = “stf_windmonster”; Value = 0x20; }
Flag { ID = “stf_windmissile”; Value = 0x40; }
Flag { ID = “stf_windany”; Value = 0x18; }
Flag { ID = “stf_tagtexmove”; Value = 0x80; }
Flag { ID = “stf_tagwind”; Value = 0x100; }
Flag { ID = “stf_floorwind”; Value = 0x200; }
Flag { ID = “stf_ceilingwind”; Value = 0x400; }
Flag { ID = “scef_player_a”; Value = 0x1; }
Flag { ID = “scef_other_a”; Value = 0x2; }
Flag { ID = “scef_monster_a”; Value = 0x4; }
Flag { ID = “scef_missile_a”; Value = 0x8; }
Flag { ID = “scef_any_a”; Value = 0x10; }
Flag { ID = “scef_ticker_a”; Value = 0x20; }
Flag { ID = “scef_player_d”; Value = 0x40; }
Flag { ID = “scef_other_d”; Value = 0x80; }
Flag { ID = “scef_monster_d”; Value = 0x100; }
Flag { ID = “scef_missile_d”; Value = 0x200; }
Flag { ID = “scef_any_d”; Value = 0x400; }
Flag { ID = “scef_ticker_d”; Value = 0x800; }
```