

ID24HACKED formal specification 0.99.1

The creation of the ID24 standard has resulted in an expansion of the capabilities of Dehacked patches. This specification details the additions and changes required to support Dehacked patches under the ID24 featureset.

Minimum engine featureset

This specification applies to any ID24 supporting source port.

Baseline features

ID24HACKED is a superset of the following Dehacked features:

- DeHackEd/Vanilla Doom
- Boom
- MUSINFO
- MBF
- DEHEXTRA
- MBF21
- DSDHACKED

It includes and supports everything defined by those ports and specifications. The above list is also considered an order, from least features to most features with each featureset being a superset of the prior one, for the sake of determining a featureset for the engine to run in.

Identifying an ID24HACKED patch

There are two primary methods for identifying an ID24HACKED patch:

- **Doom version** is set to **2024**
- Encountering anything specified in this document that is not defined in the baseline features

For the sake of convenience, a Dehacked implementation should be able to report the highest level featureset encountered so that the engine can determine whether it should consider the patch valid.

Dehacked patch initialisation order

While MBF stated that Dehacked patches from a command line are to be considered first, this behaviour conflicts with the accumulation requirements specified above. The order for Dehacked patch initialisation is now as follows:

- Every DEHACKED lump found in the WAD dictionary, in order from first loaded WAD (including the IWAD) to the last loaded WAD.
- Any .deh files encountered on the command line in the order specified (if the port supports loading from the command line).

This also applies retroactively to the previously stated baseline features.

DEHACKED limitation removals

It has become apparent that the requirements for id Software and/or its affiliates to add new features to its commercial Doom and Doom II releases and taking existing community

standards into account would result in potentially breaking any number of user mods. The following limitation removals are designed to sidestep this while allowing current and future commercial releases to add new features as required without this concern hanging overhead.

ID24HACKED builds on the **DSDHACKED** specification allowing ($2^{31} - 1$) positive indices for all supported items (frame, thing, sprite, sound). ID24HACKED extends that to allow just as many negative indices, giving a range of ($2^{32} - 1$) valid values for any index (frame, thing, sprite, sound, weapon, ammo) while also allowing custom string mnemonics starting with **USER_**.

The range of possible values is divided up amongst a few purposes, As well as limitations placed on a DEHACKED patch, those purposes are illustrated in the following table:

First index	Last index	Create?	Modify?	Purpose
0x00000000	0x7FFFFFFF	Y	Y	Original valid range.
0x80000000	0x8FFFFFFF	N	N	Implementation defined. Essentially a place for source ports to define internal objects.
0x90000000	0xBFFFFFFF	N	Y	Reserved for exclusive use by id Software and its affiliates. Any future specifications published by id Software and/or its affiliates will use indices within this range.
0xC0000000	0xFFFFFFFF	N	Y	Reserved for the community. Any future community specifications that are published should use indices from within this range.
0xFFFFFFFF	0xFFFFFFFF	N	N	Invalid index.

When dealing with 16-bit values such as the thing type number found in map definitions (specified by the **ID #** field in a Thing definition), the following values apply:

First index	Last index	Create?	Modify?	Purpose
0x0000	0x7FFF	Y	Y	Original valid range.
0x8000	0x8FFF	N	N	Implementation defined. Essentially a place for source ports to define internal objects.
0x9000	0xBFFF	N	Y	Reserved for exclusive use by id Software and its affiliates. Any future specifications published by id Software and/or its affiliates will use indices within this range.

0xC000	0xFFFE	N	Y	Reserved for the community. Any future community specifications that are published should use indices from within this range.
0xFFFF	0xFFFF	N	N	Invalid index.

The community range is not a free-use-by-anyone range. It is designed to be a range that the entire community can agree upon and implement in every source port. Any usage of this range that is not formally agreed upon by the community is considered in violation of ID24 and any and all future specifications based upon this work, and at best should be considered “in development for consideration by the community”.

String mnemonics

String mnemonics starting with **USER_** are to be added to the mnemonic resolution table. There is no limitation to what these mnemonics can be called outside of the requirement to begin with **USER_**.

In addition, the following mnemonics and their corresponding strings have been added:

Mnemonic	String
ID24_GOTINCINERATOR	You got the incinerator!
ID24_GOTCALAMITYBLADE	You got the calamity blade! Hot damn!
ID24_GOTFUELCAN	Picked up a fuel can.
ID24_GOTFUELTANK	Picked up a fuel tank.
ID24_COLOR_GREEN	Green
ID24_COLOR_INDIGO	Indigo
ID24_COLOR_BROWN	Brown
ID24_COLOR_RED	Red
ID24_COLOR_YELLOW	Yellow
ID24_COLOR_BLUE	Blue
ID24_COLOR_NAVY	Navy
ID24_COLOR_MAGENTA	Magenta

Any unknown mnemonics encountered in a Dehacked file should be ignored.

Codepointer verification

It has always been possible to specify codepointers intended for things in a frame intended for player sprites and vice versa. While it is not an expectation to catch this at parse time, it

is expected that code errors at runtime if it tries to execute a codepointer not intended for the current object.

Miscellaneous values changes

Due to the expansion of Dehacked capabilities, the following values now set values in the built-in weapon and ammo slots rather than global values:

- BFG Cells/Shot - updates the **Ammo per shot** field in the BFG weapon entry
- Initial Bullets - updates the **Initial ammo** field in the clip/bullet ammo entry

New object allocation

Note that while **DSDHACKED** specifies that indices allocate objects in a range from lowest index to highest index, this requirement is removed from **ID24HACKED**. A Dehacked patch contains perfect information allowing you to pre-determine exactly which indices are in use after a patch has been parsed. Further, allowing indices in the negative range either requires funky C array arithmetic or a very large array by reinterpreting that negative value at a bitwise level to be a positive value.

As such, **ID24HACKED** parsing only allocates objects for an index when they are encountered as a new object definition; or whenever a field that refers to those objects via an index is encountered and it was not previously defined.

One notable exception to the allocation rule is for action pointer parameters. As each value in a DeHackEd patch can be defined in any arbitrary order, to simplify parsing logic each Thing, Frame, and Sound parameter is resolved after a Frame has finished parsing.

New weapons, ammo types, things, frames, sprites, and sounds

A complete table of all new additions to the internal tables can be found in <a separate file stored with these specifications>. These tables must be compiled and constructed into a master list - known as the **in-order table** - and inserted into the corresponding **associative map** before allocating new things from a DeHackEd patch.

Table construction

Each **in-order table** has a related **associative map** used for resolving objects via an index. These **associative maps** are to be used in place of the existing tables. **Associative maps** exist for each of the following datatypes:

- Thing
- Frame
- Sprite
- Sound
- Weapon
- Ammo

An extra **associative map** is maintained for the **ID #** field of a Thing definition (referred to as the **spawn map**). Spawning a thing is now resolved from the **spawn map** instead of iterating through the **in-order** table.

The **in-order tables** are constructed in the following way for every data type except things:

- Original built-in tables in order
- ID24 tables in order

Things are constructed the following way:

- Original built-in tables in order
- MUSINFO 14101-14064 entries in order
- ID24 tables in order

Any new entries defined by the DeHackedPatch must be placed after the entries as described above and inserted into the **associative map**.

At program initialisation and after any DeHacked patch is loaded, each **associative map** must be rebuilt with the following logic:

- Clear every **associative map**
- For each **in-order table**
 - Create a copy and sort each **item** in ascending order by its identifying index
 - If the **item** is a Thing, sort any **item** with matching identifying indices by **ID #**
 - If the related **associative map** is the **spawn map**, do not sort the **in-order table** at all
 - Iterate through each **item** in the **in-order table**
 - If this **item** does not exist in the related **associative map**, then insert it into the **associative map**

New objects are to be pushed to the back of the **in-order tables**; and also added to the **associative map** immediately in order for DeHacked patches to correctly refer to objects defined earlier in the patch. Note that while the order is not important during a DeHacked patch, it is important for the game simulation to maintain backwards compatibility with vanilla (in particular, the **spawn map** rules are designed to work . You don't want to skip the **associative map** reconstruction step.

Accumulating Dehacked patches

While it has always been possible to load DeHacked patches on top of one another, at best this has always been undefined behaviour. DeHacked as originally specified has an expectation that it is operating on a vanilla Doom set of tables. It makes no attempts to verify if this is true.

Due to needing to avoid undefined behaviour for the sake of running on video game console platforms, DeHacked patches can now define a series of hash values that are calculated from the tables before any given DeHacked patch is applied. These hash values indicate that a Dehacked patch has been tested and confirmed to work with previously loaded DeHacked patches. This is applied retroactively to every previous DeHacked specification - an **ID24** capable port can consider and process this regardless of the feature level a mod requests/supports and is preferred to be the default way to handle DeHacked patches (with it being an outright requirement in the official versions of Doom and Doom II sold in stores of all kinds).

As this is a lengthy topic, further information on hashing is provided in a separate document.

Data types

Each data type used in the tables following this section correspond to the following:

- **string** - C-style null-terminated string, stored as a pointer
- **bitfield** - 32-bit integer
- **integer** - 32-bit signed integer
- **enum** - 32-bit signed integer
- **bool** - 8-bit integer
- **fixed** - 32-bit signed integer

Frame additions

A frame can now be rendered with a transparency lump. This lump applies to both thing and player sprite states.

IDHACKED24 adds the following parameters to a Frame definition:

Name	Type	Default	Description
Tranmap	string	null	The name of a transparency map lump to use when rendering the sprite associated with this frame. Note: While a Thing can use the built-in transparency map with the TRANSLUCENT flag, a frame's Tranmap will override this.

Frame defaults

All frames defined by prior specifications are to have default values set to those defined in the fields table; all used-defined things likewise will have those same default values.

Thing additions

Things have had a sizable expansion of functionality in **IDHACKED24**.

Things now have some capacity to control their behaviour when respawning monsters is turned on (either via command line or the Nightmare! difficulty setting). They can control if they're allowed to respawn, as well as how long they must stay dead at a minimum and their chances of respawning.

Special items previously had the ability to remain in the world on collection as a hardcoded feature of certain multiplayer modes. A thing is now able to explicitly define this behavior for single player, cooperative, and deathmatch modes.

Dropped items are no longer hardcoded to Thing type. Any thing is able to define a thing index representing the item to drop on death. Note that when a source port is not operating with **ID24** compatibility that vanilla behaviour must be retained.

Special item collection is no longer hardcoded to sprite names. As such, a full suite of values to handle collection is exposed. When a thing does not define any of the **ID24** values for item collection with the exception of the **Pickup message**, the vanilla behaviour is retained. Any

item with a **Pickup message** overwrites the defined vanilla message regardless of behavior.

Note that when a source port is not operating with **ID24** compatibility, vanilla behaviour must be retained.

IDHACKED24 adds the following fields to a Thing definition:

Name	Type	Default	Description
ID24 Bits	bitfield	0	New flags to control ID24 thing features.
Min respawn tics	integer	420	The number of tics to wait when respawning monsters is turned on before attempting to respawn.
Respawn dice	integer	4	The value that a RNG value (between 0 and 255) must be greater than to allow this item to respawn.
Dropped item	integer	-1	The thing ID to spawn on death.
Pickup ammo type	integer	-1	The ammo ID to pick up when collecting this SPECIAL thing.
Pickup ammo category	bitfield	-1	The ammo category to resolve a quantity from when collecting this SPECIAL thing.
Pickup weapon type	integer	-1	The weapon ID to pick up when collecting this SPECIAL thing.
Pickup item type	enum	-1	The powerup to pick up when collecting this SPECIAL thing.
Pickup bonus count	integer	6	A value to add to the screen flash counter when collecting this SPECIAL thing.
Pickup sound	integer	0	The sound ID to play when collecting this SPECIAL thing.
Pickup message	string	null	The string mnemonic to resolve and display when picking up this SPECIAL thing.
Translation	string	null	The translation lump to use when rendering this thing.

ID24 bits

The following values apply to the **ID24 bits** bitfield (with mnemonics specified in [] brackets) and are allowed to be combined with any other value:

- 1 [NORESPAWN] - Does not respawn when respawning monsters is turned on
- 2 [SPECIALSTAYSSINGLE] - Special remains in the world when collected in single player mode

- 4 [SPECIALSTAYSCOOP] - Special remains in the world when collected in cooperative multiplayer mode
- 8 [SPECIALSTAYSDM] - Special remains in the world when collected in deathmatch multiplayer mode

Pickup ammo category

The following values apply to the **Pickup ammo category** bitfield and are exclusive to one another:

- 0 - clip ammo
- 1 - box ammo
- 2 - weapon ammo
- 3 - backpack ammo

The following values apply to the **Pickup ammo category** bitfield and are allowed to be combined with any other value:

- 4 - dropped
- 8 - deathmatch

A value of -1 in the **Pickup ammo category** bitfield means that there is no category and overrides any bit set as described above.

Pickup item type

The following values apply to the **Pickup item type** enumeration:

- -1 - no item
- 0 - message only
- 1 - blue keycard
- 2 - yellow keycard
- 3 - red keycard
- 4 - blue skull
- 5 - yellow skull
- 6 - red skull
- 7 - backpack
- 8 - health bonus
- 9 - stimpack
- 10 - medikit
- 11 - soulsphere
- 12 - megasphere
- 13 - armor bonus
- 14 - green armor
- 15 - blue armor
- 16 - computer area map
- 17 - light amplification goggles
- 18 - berserk
- 19 - partial invisibility
- 20 - radiation shielding suit
- 21 - invulnerability

Thing defaults

All things defined by prior specifications are to have default values set to those defined in the fields table; all used-defined things likewise will have those same default values.

However, the MF_TRANSLUCENT flag added to select things by Boom tables is to be removed from the tables entirely. The flag must still be allowed to be set by a DeHackEd patch, but the default tables must reflect vanilla Doom for all relevant values.

Some exceptional default values must be set on certain hardcoded things. These are:

Thing ID	Field	Value
MT_MISC4	ID24 Bits	SPECIALSTAYSCOOP
MT_MISC5	ID24 Bits	SPECIALSTAYSCOOP
MT_MISC6	ID24 Bits	SPECIALSTAYSCOOP
MT_MISC7	ID24 Bits	SPECIALSTAYSCOOP
MT_MISC8	ID24 Bits	SPECIALSTAYSCOOP
MT_MISC9	ID24 Bits	SPECIALSTAYSCOOP
MT_POSSESSED	Dropped item	MT_CLIP
MT_SHOTGUY	Dropped item	MT_SHOTGUN
MT_CHAINGUY	Dropped item	MT_CHAINGUN
MT_WOLFSS	Dropped item	MT_CLIP

Weapon additions

It is now allowed to define weapons not previously defined by the built-in tables.

Weapons can now define which slot they live in, as well as the priority for selection when pressing the key for that slot. The weapon with the highest slot priority in any given slot will be selected first when activating that slot; subsequent activations will descend down the list of weapons for that slot in decreasing priority.

Weapons can also define their place in the autoswitch priority list. When autoswitching is activated, the weapon with the highest priority will be considered first and will then descend down the list of weapons in decreasing priority.

Weapons can now define whether they are in the player's inventory on respawn, as well as which weapon should be the first one raised. Note that in the case of multiple weapons defining themselves as the first one raised, only the last one encountered in declaration order will be raised.

Carousel icons are an optional feature used by the official releases of Doom and Doom II, primarily to assist with weapon selection on a control pad. It is entirely at a port's discretion if

it implements this feature; however, a port must still parse and set all carousel fields correctly regardless.

The original Doom disallowed selecting the fist weapon when a chainsaw was owned and a berserk pack was not picked up in the current level. To replicate - and expand upon - this ability, a few additional fields with the following logic have been included:

- You start being allowed to select this weapon if you own it
- If **No switch with owned weapon** is defined and you own that weapon, you are disallowed from selecting this weapon
- If **Allow switch with owned weapon** is defined and you own that weapon, you are allowed to select this weapon
- If you are allowed to select this weapon *and* if **No switch with owned item** is defined and you own that item, you are disallowed to select this weapon
- If you are disallowed to select this weapon *and* if **Allow switch with owned item** is defined and you own that item, you are allowed to select this weapon
- If you are still allowed to select this weapon, select this weapon

To resolve the above logic, the weapon index resolves via the weapon lookup table; and the item index resolves via the table described in “Pickup item type”.

Note that **wp_nochange** must be redefined to -1 in code to be compliant with the above index range definitions.

IDHACKED24 adds the following parameters to a weapon definition:

Name	Type	Default	Description
Slot	integer	-1	Which slot to bind this weapon to.
Slot Priority	integer	-1	Priority value for selection in this slot.
Switch Priority	integer	-1	Priority value when autoswitching.
Initial Owned	bool	false	Whether this weapon is available to the player on respawn.
Initial Raised	bool	false	Whether this weapon is the one to be raised on respawn.
Carousel icon	string	“SMUNKN”	A patch to be used as a small icon for weapon selection wheels/carousels/etc.
Allow switch with owned weapon	integer	-1	Allow weapon switching according to described logic.
No switch with owned weapon	integer	-1	Disallow weapon switching according to described logic.

Allow switch with owned item	integer	-1	Allow weapon switching according to described logic.
No switch with owned item	integer	-1	Disallow weapon switching according to described logic.

Weapon defaults

All used-defined weapons will have defaults set corresponding to the above table in addition to all defaults for previous specifications.

For all built-in weapons, the following values must be set in addition to all defaults for previous specifications:

Weapon	Field	Value
wp_fist	Slot	1
	Slot Priority	0
	Switch Priority	0
	Initial Owned	true
	Initial Raised	false
	Carousel icon	"SMFIST"
wp_pistol	Slot	2
	Slot Priority	0
	Switch Priority	6
	Initial Owned	true
	Initial Raised	true
	Carousel icon	"SMPISG"
wp_shotgun	Slot	3
	Slot Priority	0
	Switch Priority	7
	Initial Owned	false
	Initial Raised	false
	Carousel icon	"SMSHOT"
wp_chaingun	Slot	4

	Slot Priority	0
	Switch Priority	8
	Initial Owned	false
	Initial Raised	false
	Carousel icon	"SMMGUN"
wp_missile	Slot	5
	Slot Priority	0
	Switch Priority	4
	Initial Owned	false
	Initial Raised	false
	Carousel icon	"SMLAUN"
wp_plasma	Slot	6
	Slot Priority	0
	Switch Priority	10
	Initial Owned	false
	Initial Raised	false
	Carousel icon	"SMPLAS"
wp_bfg	Slot	7
	Slot Priority	0
	Switch Priority	2
	Initial Owned	false
	Initial Raised	false
	Carousel icon	"SMBFGG"
wp_chainsaw	Slot	1
	Slot Priority	1
	Switch Priority	5
	Initial Owned	false
	Initial Raised	false
	Carousel icon	"SMCSAW"

wp_supershotgun	Slot	3
	Slot Priority	1
	Switch Priority	9
	Initial Owned	false
	Initial Raised	false
	Carousel icon	"SMSGN2"

Ammo additions

It is now allowed to define ammo types not previously defined by the built-in tables.

Every aspect of an ammo type is now customisable, and does not rely on the vanilla behaviour of multiplying ammo values to determine how much ammo is in a box, a weapon, or a backpack.

The skill multiplier values can now be defined independently for each skill, and round down the resulting value to get a whole integer value.

Note that **am_noammo** must be redefined to -1 in code to be compliant with the above index range definitions.

IDHACKED24 adds the following parameters to an ammo definition:

Name	Type	Default	Description
Initial ammo	integer	0	How much of this ammo the player receives on respawn.
Max upgraded ammo	integer	0	The value that the maximum amount of ammo is set to on collecting a backpack.
Box ammo	integer	0	How much ammo to receive when collecting a box with this ammo type.
Backpack ammo	integer	0	How much ammo to receive when collecting a backpack.
Weapon ammo	integer	0	How much ammo to receive when collecting a weapon with this ammo type.
Dropped ammo	integer	0	How much ammo to receive when collecting a dropped clip with this ammo type.
Dropped box ammo	integer	0	How much ammo to receive when collecting a dropped box with this ammo type.
Dropped backpack ammo	integer	0	How much ammo to receive when collecting a dropped backpack.

Dropped weapon ammo	integer	0	How much ammo to receive when collecting a dropped weapon with this ammo type.
Deathmatch weapon ammo	integer	0	How much ammo to receive when collecting a weapon with this ammo type in deathmatch modes.
Skill 1 multiplier	fixed	131072 (2.0)	The multiplier to apply to all collected ammo counts on skill 1 (I'm Too Young To Die)
Skill 2 multiplier	fixed	65536 (1.0)	The multiplier to apply to all collected ammo counts on skill 2 (Hey, Not Too Rough)
Skill 3 multiplier	fixed	65536 (1.0)	The multiplier to apply to all collected ammo counts on skill 3 (Hurt Me Plenty)
Skill 4 multiplier	fixed	65536 (1.0)	The multiplier to apply to all collected ammo counts on skill 4 (Ultra-Violence)
Skill 5 multiplier	fixed	131072 (2.0)	The multiplier to apply to all collected ammo counts on skill 5 (Nightmare!)

Per ammo and Max ammo

If the only fields set in a weapon entry are **Per ammo** and/or **Max ammo**, then a Dehacked parser is expected to fill out the the following fields in the a manner consistent with vanilla ammo calculations (ie integer operations, meaning all divides are rounded down), in order:

Value	New value
Max upgraded ammo	Max ammo * 2
Box ammo	Per ammo * 5
Backpack ammo	Per ammo
Weapon ammo	Per ammo * 2
Dropped ammo	Per ammo / 2
Dropped box ammo	Box ammo / 2
Dropped backpack ammo	Backpack ammo / 2
Dropped weapon ammo	Weapon ammo / 2
Deathmatch weapon ammo	Per ammo * 5

Note that this logic also applies to accumulative Dehacked patches. Whether an ammo definition has previously been created/modified by Dehacked (or is from an internal table) is not considered. As such, ammo definitions that expect to be accumulative in ID24 will work best by explicitly providing every value required.