DFHack Documentation

Release 0.42.06-

The DFHack Team

April 24, 2016

Contents

CHAPTER 1

Introduction

DFHack is a Dwarf Fortress memory access library, distributed with a wide variety of useful scripts and plugins.

The project is currently hosted at https://www.github.com/DFHack/dfhack, and can be downloaded from the releases page.

All new releases are announced in the bay12 forums thread, which is also a good place for discussion and questions.

For users, it provides a significant suite of bugfixes and interface enhancements by default, and more can be enabled. There are also many tools (such as *workflow* or *autodump*) which can make life easier. You can even add third-party scripts and plugins to do almost anything!

For modders, DFHack makes many things possible. Custom reactions, new interactions, magic creature abilities, and more can be set through *Scripts for Modders* and custom raws. Non-standard DFHack scripts and inits can be stored in the raw directory, making raws or saves fully self-contained for distribution - or for coexistence in a single DF install, even with incompatible components.

For developers, DFHack unites the various ways tools access DF memory and allows easier development of new tools. As an open-source project under *various copyleft licences*, contributions are welcome.

CHAPTER 2

User Manual

2.1 Introduction and Overview

DFHack is a Dwarf Fortress memory access library, distributed with a wide variety of useful scripts and plugins.

The project is currently hosted at https://www.github.com/DFHack/dfhack, and can be downloaded from the releases page.

All new releases are announced in the bay12 forums thread, which is also a good place for discussion and questions.

For users, it provides a significant suite of bugfixes and interface enhancements by default, and more can be enabled. There are also many tools (such as *workflow* or *autodump*) which can make life easier. You can even add third-party scripts and plugins to do almost anything!

For modders, DFHack makes many things possible. Custom reactions, new interactions, magic creature abilities, and more can be set through *Scripts for Modders* and custom raws. Non-standard DFHack scripts and inits can be stored in the raw directory, making raws or saves fully self-contained for distribution - or for coexistence in a single DF install, even with incompatible components.

For developers, DFHack unites the various ways tools access DF memory and allows easier development of new tools. As an open-source project under *various copyleft licences*, contributions are welcome.

Contents

2.1.1 Installing DFHack

DFHack is available for the SDL version of Dwarf Fortress on Windows, any modern Linux distribution, and Mac OS X (10.6.8 and later). It is possible to use Windows DF+DFHack under Wine on Linux or OS X.

Most releases only support the version of DF mentioned in their title - for example, DFHack 0.40.24-r2 only supports DF 0.40.24 - but some releases support earlier DF versions as well. Wherever possible, use the latest version of DFHack built for the target version of DF.

Installing DFhack involves copying files from a release archive into your DF folder, so that:

- On Windows, `SDL.dll` is replaced

- On Linux or OSX, the `dfhack` script is placed in the same folder as the `df` script

Uninstalling is basically the same, in reverse:

- On Windows, replace `SDL.dll` with `SDLreal.dll`, then remove the DFHack files.
- On Linux or OSX, remove the DFHack files.

New players may wish to get a pack with DFHack preinstalled.

2.1.2 Getting started

DFHack basically extends DF with something similar to the console found in many PC games.

If DFHack is installed correctly, it will automatically pop up a console window once DF is started as usual on Windows. Linux and Mac OS X require running the dfhack script from the terminal, and will use that terminal for the console.

- Basic interaction with dfhack involves entering commands into the console. To learn what commands are available, you can keep reading this documentation or skip ahead and use the *ls* and *help* commands.
- Another way to interact with DFHack is to set in-game *keybindings* for certain commands. Many of the newer and user-friendly tools are designed to be used this way.
- Commands can also run at startup via *init files*, on in batches at other times with the *script* command.
- Finally, some commands are persistent once enabled, and will sit in the background managing or changing some aspect of the game if you *enable* them.

2.1.3 Troubleshooting

Don't panic! Even if you need this section, it'll be OK :)

If something goes wrong, check the log files in DF's folder (`stderr.log` and `stdout.log`). Looking at these might help you - or someone else - solve the problem. Take screenshots of any weird error messages, and take notes on what you did to cause them.

If the search function in this documentation isn't enough and the DF Wiki hasn't helped, try asking in:

- the #dfhack IRC channel on freenode
- the Bay12 DFHack thread
- the /r/dwarffortress questions thread
- the thread for the mod or Starter Pack you're using (if any)

2.2 DFHack Core

DFHack commands can be implemented in three ways, all of which are used in the same way:

builtin commands are implemented by the core of DFHack. They manage other DFhack tools, interpret commands, and control basic aspects of DF (force pause or quit).

plugins are stored in `hack/plugins/` and must be compiled with the same version of DFHack. They are less flexible than scripts, but used for complex or ongoing tasks becasue they run faster.

scripts are Ruby or Lua scripts stored in `hack/scripts/`. Because they don't need to be compiled, scripts are more flexible about versions, and easier to distribute. Most third-party DFHack addons are scripts.

Contents

2.2.1 Built-in Commands

The following commands are provided by the 'core' components of DFhack, rather than plugins or scripts.

cls

Clear the terminal. Does not delete command history.

die

Instantly kills DF without saving.

enable

Many plugins can be in a distinct enabled or disabled state. Some of them activate and deactivate automatically depending on the contents of the world raws. Others store their state in world data. However a number of them have to be enabled globally, and the init file is the right place to do it.

Most such plugins or scripts support the built-in `enable` and `disable` commands. Calling them at any time without arguments prints a list of enabled and disabled plugins, and shows whether that can be changed through the same commands.

To enable or disable plugins that support this, use their names as arguments for the command:

```
enable manipulator search
```

fpause

Forces DF to pause. This is useful when your FPS drops below 1 and you lose control of the game.

help

Most commands support using the `help <command>` built-in command to retrieve further help without having to look at this document. `? <cmd>` and `man <cmd>` are aliases.

Some commands (including many scripts) instead take `help` or `?` as an option on their command line - ie `<cmd> help`.

hide

Hides the DFHack terminal window. Only available on Windows.

keybinding

To set keybindings, use the built-in `keybinding` command. Like any other command it can be used at any time from the console, but bindings are not remembered between runs of the game unless re-created in *dfhack*.init*.

Currently, any combinations of Ctrl/Alt/Shift with A-Z, 0-9, or F1-F12 are supported.

Possible ways to call the command:

`keybinding list <key>` List bindings active for the key combination.

`keybinding clear <key> <key>...` Remove bindings for the specified keys.

`keybinding add <key> "cmdline" "cmdline"...` Add bindings for the specified key.

`keybinding set <key> "cmdline" "cmdline"...` Clear, and then add bindings for the specified key.

The `<key>` parameter above has the following *case-sensitive* syntax:

```
[Ctrl-][Alt-][Shift-]KEY[@context[|context...]]
```

where the *KEY* part can be any recognized key and [] denote optional parts.

When multiple commands are bound to the same key combination, DFHack selects the first applicable one. Later `add` commands, and earlier entries within one `add` command have priority. Commands that are not specifically intended for use as a hotkey are always considered applicable.

The `context` part in the key specifier above can be used to explicitly restrict the UI state where the binding would be applicable. If called without parameters, the `keybinding` command among other things prints the current context string.

Only bindings with a `context` tag that either matches the current context fully, or is a prefix ending at a `/` boundary would be considered for execution, i.e. when in context `foo/bar/baz`, keybindings restricted to any of `@foo/bar/baz`, `@foo/bar`, `@foo` or none will be active.

Multiple contexts can be specified by separating them with a pipe (`|`) - for example, `@foo|bar|baz/foo` would match anything under `@foo`, `@bar`, or `@baz/foo`.

Interactive commands like *liquids* cannot be used as hotkeys.

kill-lua

Stops any currently-running Lua scripts. By default, scripts can only be interrupted every 256 instructions. Use `kill-lua force` to interrupt the next instruction.

load

`load`, `unload`, and `reload` control whether a plugin is loaded into memory - note that plugins are loaded but disabled unless you do something. Usage:

```
load|unload|reload PLUGIN|(-a|--all)
```

Allows dealing with plugins individually by name, or all at once.

ls

`ls` does not list files like the Unix command, but rather available commands - first built in commands, then plugins, and scripts at the end. Usage:

ls -a Also list scripts in subdirectories of `hack/scripts/`, which are generally not intended for direct use.

ls <plugin> List subcommands for the given plugin.

plug

Lists available plugins, including their state and detailed description.

`plug` Lists available plugins (*not* commands implemented by plugins)

`plug [PLUGIN] [PLUGIN] ...` List state and detailed description of the given plugins, including commands implemented by the plugin.

sc-script

Allows additional scripts to be run when certain events occur (similar to onLoad*.init scripts)

script

Reads a text file, and runs each line as a DFHack command as if it had been typed in by the user - treating the input like *an init file*.

Some other tools, such as *autobutcher* and *workflow*, export their settings as the commands to create them - which are later loaded with `script`

show

Shows the terminal window after it has been *hidden*. Only available on Windows. You'll need to use it from a *keybinding* set beforehand, or the in-game *command-prompt*.

type

`type command` shows where `command` is implemented.

Other Commands

The following commands are *not* built-in, but offer similarly useful functions.

- *command-prompt*
- *hotkeys*
- *lua*
- *multicmd*
- *nopause*
- *quicksave*
- *rb*
- *repeat*

2.2.2 Init Files

DFHack allows users to automatically run commonly-used DFHack commands when DF is first loaded, when a game is loaded, and when a game is unloaded.

Init scripts function the same way they would if the user manually typed in their contents, but are much more convenient. In order to facilitate savegave portability, mod merging, and general organization of init files, DFHack supports multiple init files both in the main DF directory and save-specific init files in the save folders.

DFHack looks for init files in three places each time they could be run:

1. The main DF directory
2. `data/save/world/raw`, where `world` is the current save, and
3. `data/save/world/raw/objects`

When reading commands from dfhack.init or with the *script* command, if the final character on a line is a backslash then the next uncommented line is considered a continuation of that line, with the backslash deleted. Commented lines are skipped, so it is possible to comment out parts of a command with the # character.

dfhack*.init

If your DF folder contains at least one file named `dfhack*.init` (where `*` is a placeholder for any string), then all such files are executed in alphabetical order when DF is first started.

DFHack is distributed with `/dfhack.init-example` as an example with an up-to-date collection of basic commands; mostly setting standard keybindings and *enabling* plugins. You are encouraged to look through this file to learn which features it makes available under which key combinations. You may also customise it and rename it to `dfhack.init`.

If your DF folder does not contain any `dfhack*.init` files, the example will be run as a fallback.

These files are best used for keybindings and enabling persistent plugins which do not require a world to be loaded.

onLoad*.init

When a world is loaded, DFHack looks for files of the form `onLoad*.init`, where `*` can be any string, including the empty string.

All matching init files will be executed in alphebetical order. A world being loaded can mean a fortress, an adventurer, or legends mode.

These files are best used for non-persistent commands, such as setting a *fix* script to run on *repeat*.

onUnload*.init

When a world is unloaded, DFHack looks for files of the form `onUnload*.init`. Again, these files may be in any of the above three places. All matching init files will be executed in alphebetical order.

Modders often use such scripts to disable tools which should not affect an unmodded save.

Other init files

- `onMapLoad*.init` and `onMapUnload*.init` are run when a map, distinct from a world, is loaded. This is good for map-affecting commands (eg *clean*), or avoiding issues in Legends mode.
- Any lua script named `raw/init.d/*.lua`, in the save or main DF directory, will be run when any world or that save is loaded.

2.2.3 Miscellaneous Notes

This section is for odd but important notes that don't fit anywhere else.

- The `dfhack-run` executable is there for calling DFHack commands in an already running DF+DFHack instance from external OS scripts and programs, and is *not* the way how you use DFHack normally.
- If a DF `H` hotkey is named with a DFHack command, pressing the corresponding `Fx` button will run that command, instead of zooming to the set location.
- The command line has some nice line editing capabilities, including history that's preserved between different runs of DF (use up/down keys to go through the history).
- The binaries for 0.40.15-r1 to 0.34.11-r4 are on DFFD. Older versions are available here.
- To include whitespace in the argument/s to some command, quote it in double quotes. To include a double quote character, use `\"`.
- If the first non-whitespace character is `:`, the command is parsed in an alternative mode which is very useful for the *lua* and *rb* commands. The following two command lines are exactly equivalent:

```
:foo a b "c d" e f
foo "a b \"c d\" e f"
```

  - non-whitespace characters following the `:` are the command name
  - the remaining part of the line is used verbatim as the first argument

2.3 DFHack Plugins

DFHack plugins are the commands, that are compiled with a specific version. They can provide anything from a small keybinding, to a complete overhaul of game subsystems or the entire renderer.

Most commands offered by plugins are listed here, hopefully organised in a way you will find useful.

Contents

2.3.1 Data inspection and visualizers

stonesense

An isometric visualizer that runs in a second window. Usage:

stonesense Open the visualiser in a new window. Alias `ssense`.

ssense overlay Overlay DF window, replacing the map area.

For more information, see the full Stonesense README.

blueprint

Exports a portion of your fortress into QuickFort style blueprint files.:

```
blueprint <x> <y> <z> <name> [dig] [build] [place] [query]
```

Options (If only region and name are given, export all):

x,y,z Size of map area to export

name Name of export files

dig Export dig commands to "<name>-dig.csv"

build Export build commands to "<name>-build.csv"

place Export stockpile commands to "<name>-place.csv"

query Export query commands to "<name>-query.csv"

Goes very well with *fortplan*, for re-importing.

remotefortressreader

An in-development plugin for realtime fortress visualisation. See Armok Vision.

cursecheck

Checks a single map tile or the whole map/world for cursed creatures (ghosts, vampires, necromancers, werebeasts, zombies).

With an active in-game cursor only the selected tile will be observed. Without a cursor the whole map will be checked.

By default cursed creatures will be only counted in case you just want to find out if you have any of them running around in your fort. Dead and passive creatures (ghosts who were put to rest, killed vampires, ...) are ignored. Undead skeletons, corpses, bodyparts and the like are all thrown into the curse category "zombie". Anonymous zombies and resurrected body parts will show as "unnamed creature".

Options:

detail Print full name, date of birth, date of curse and some status info (some vampires might use fake identities in-game, though).

nick Set the type of curse as nickname (does not always show up in-game, some vamps don't like nicknames).

all Include dead and passive cursed creatures (can result in a quite long list after having FUN with necromancers).

verbose Print all curse tags (if you really want to know it all).

Examples:

`cursecheck detail all` Give detailed info about all cursed creatures including deceased ones (no in-game cursor).

`cursecheck nick` Give a nickname all living/active cursed creatures on the map(no in-game cursor).

Note: If you do a full search (with the option "all") former ghosts will show up with the cursetype "unknown" because their ghostly flag is not set.

Please report any living/active creatures with cursetype "unknown" - this is most likely with mods which introduce new types of curses.

---

flows

A tool for checking how many tiles contain flowing liquids. If you suspect that your magma sea leaks into HFS, you can use this tool to be sure without revealing the map.

probe

Can be used to determine tile properties like temperature.

prospect

Prints a big list of all the present minerals and plants. By default, only the visible part of the map is scanned.

Options:

all Scan the whole map, as if it was revealed.

value Show material value in the output. Most useful for gems.

hell Show the Z range of HFS tubes. Implies 'all'.

If prospect is called during the embark selection screen, it displays an estimate of layer stone availability.

Note: The results of pre-embark prospect are an *estimate*, and can at best be expected to be somewhere within +/- 30% of the true amount; sometimes it does a lot worse. Especially, it is not clear how to precisely compute how many soil layers there will be in a given embark tile, so it can report a whole extra layer, or omit one that is actually present.

Options:

all Also estimate vein mineral amounts.

reveal

This reveals the map. By default, HFS will remain hidden so that the demons don't spawn. You can use `reveal hell` to reveal everything. With hell revealed, you won't be able to unpause until you hide the map again. If you really want to unpause with hell revealed, use `reveal demons`.

Reveal also works in adventure mode, but any of its effects are negated once you move. When you use it this way, you don't need to run `unreveal`.

Usage and related commands:

reveal Reveal the whole map, except for HFS to avoid demons spawning

reveal hell Also show hell, but requires `unreveal` before unpausing

reveal demon Reveals everything and allows unpausing - good luck!

unreveal Reverts the effects of `reveal`

revtoggle Switches between `reveal` and `unreveal`

revflood Hide everything, then reveal tiles with a path to the cursor (useful to make walled-off rooms vanish)

revforget Discard info about what was visible before revealing the map. Only useful where (eg) you abandoned with the fort revealed and no longer want the data.

showmood

Shows all items needed for the currently active strange mood.

2.3.2 Bugfixes

fixdiplomats

Adds a Diplomat position to all Elven civilizations, allowing them to negotiate tree cutting quotas - and you to violate them and start wars. This was vanilla behaviour until `0.31.12`, in which the "bug" was "fixed".

fixmerchants

Adds the Guild Representative position to all Human civilizations, allowing them to make trade agreements. This was the default behaviour in `0.28.181.40d` and earlier.

fix-unit-occupancy

This plugin fixes issues with unit occupancy, notably phantom "unit blocking tile" messages (Bug 3499). It can be run manually, or periodically when enabled with the built-in enable/disable commands:

(no argument) Run the plugin once immediately, for the whole map.

-h, here, cursor Run immediately, only operate on the tile at the cursor

-n, dry, dry-run Run immediately, do not write changes to map

interval <X> Run the plugin every `X` ticks (when enabled). The default is 1200 ticks, or 1 day. Ticks are only counted when the game is unpaused.

fixveins

Removes invalid references to mineral inclusions and restores missing ones. Use this if you broke your embark with tools like *tiletypes*, or if you accidentally placed a construction on top of a valuable mineral floor.

petcapRemover

Allows you to remove or raise the pet population cap. In vanilla DF, pets will not reproduce unless the population is below 50 and the number of children of that species is below a certain percentage. This plugin allows removing the second restriction and removing or raising the first. Pets still require PET or PET_EXOTIC tags in order to reproduce. Type `help petcapRemover` for exact usage. In order to make population more stable and avoid sudden population booms as you go below the raised population cap, this plugin counts pregnancies toward the new population cap. It can still go over, but only in the case of multiple births.

Usage:

petcapRemover cause pregnancies now and schedule the next check

petcapRemover every n set how often in ticks the plugin checks for possible pregnancies

petcapRemover cap n set the new cap to n. if n = 0, no cap

petcapRemover pregtime n sets the pregnancy duration to n ticks. natural pregnancies are 300000 ticks for the current race and 200000 for everyone else

tweak

Contains various tweaks for minor bugs.

One-shot subcommands:

clear-missing Remove the missing status from the selected unit. This allows engraving slabs for ghostly, but not yet found, creatures.

clear-ghostly Remove the ghostly status from the selected unit and mark it as dead. This allows getting rid of bugged ghosts which do not show up in the engraving slab menu at all, even after using clear-missing. It works, but is potentially very dangerous - so use with care. Probably (almost certainly) it does not have the same effects like a proper burial. You've been warned.

fixmigrant Remove the resident/merchant flag from the selected unit. Intended to fix bugged migrants/traders who stay at the map edge and don't enter your fort. Only works for dwarves (or generally the player's race in modded games). Do NOT abuse this for 'real' caravan merchants (if you really want to kidnap them, use 'tweak makeown' instead, otherwise they will have their clothes set to forbidden etc).

makeown Force selected unit to become a member of your fort. Can be abused to grab caravan merchants and escorts, even if they don't belong to the player's race. Foreign sentients (humans, elves) can be put to work, but you can't assign rooms to them and they don't show up in DwarfTherapist because the game treats them like pets. Grabbing draft animals from a caravan can result in weirdness (animals go insane or berserk and are not flagged as tame), but you are allowed to mark them for slaughter. Grabbing wagons results in some funny spam, then they are scuttled.

Subcommands that persist until disabled or DF quits:

adamantine-cloth-wear Prevents adamantine clothing from wearing out while being worn (Bug 6481).

advmode-contained Works around Bug 6202, custom reactions with container inputs in advmode. The issue is that the screen tries to force you to select the contents separately from the container. This forcefully skips child reagents.

block-labors Prevents labors that can't be used from being toggled

civ-view-agreement Fixes overlapping text on the "view agreement" screen

craft-age-wear Fixes the behavior of crafted items wearing out over time (Bug 6003). With this tweak, items made from cloth and leather will gain a level of wear every 20 years.

embark-profile-name Allows the use of lowercase letters when saving embark profiles

eggs-fertile Displays a fertility indicator on nestboxes

farm-plot-select Adds "Select all" and "Deselect all" options to farm plot menus

fast-heat Further improves temperature update performance by ensuring that 1 degree of item temperature is crossed in no more than specified number of frames when updating from the environment temperature. This reduces the time it takes for stable-temp to stop updates again when equilibrium is disturbed.

fast-trade Makes Shift-Down in the Move Goods to Depot and Trade screens select the current item (fully, in case of a stack), and scroll down one line.

fps-min Fixes the in-game minimum FPS setting

hide-priority Adds an option to hide designation priority indicators

import-priority-category Allows changing the priority of all goods in a category when discussing an import agreement with the liaison

kitchen-keys Fixes DF kitchen meal keybindings (Bug 614)

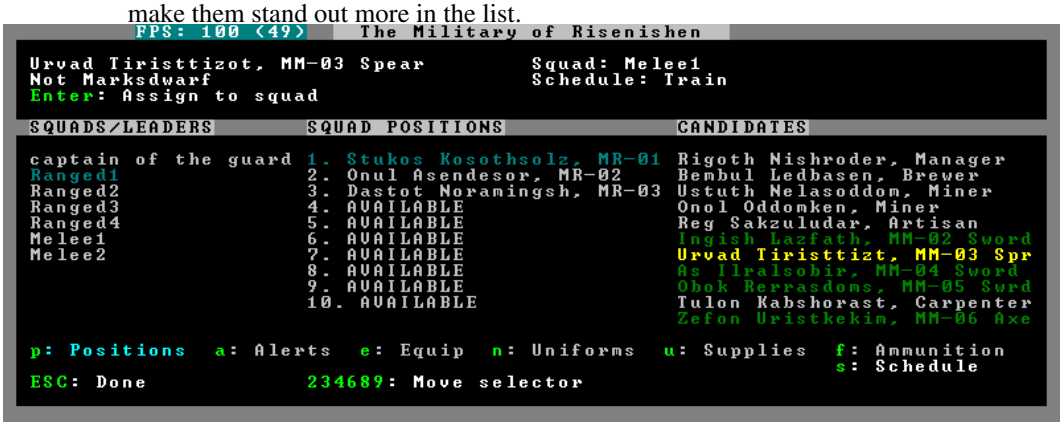
kitchen-prefs-color Changes color of enabled items to green in kitchen preferences

kitchen-prefs-empty Fixes a layout issue with empty kitchen tabs (Bug 9000)

manager-quantity Removes the limit of 30 jobs per manager order

max-wheelbarrow Allows assigning more than 3 wheelbarrows to a stockpile

military-color-assigned Color squad candidates already assigned to other squads in yellow/green to make them stand out more in the list.



military-stable-assign Preserve list order and cursor position when assigning to squad, i.e. stop the rightmost list of the Positions page of the military screen from constantly resetting to the top.

nestbox-color Fixes the color of built nestboxes

shift-8-scroll Gives Shift-8 (or *) priority when scrolling menus, instead of scrolling the map

stable-cursor Saves the exact cursor position between t/q/k/d/b/etc menus of fortress mode.

title-start-rename Adds a safe rename option to the title screen "Start Playing" menu

tradereq-pet-gender Displays pet genders on the trade request screen

fix-armory

This plugin requires a binpatch, which has not been available since DF 0.34.11

2.3.3 UI Upgrades

Note: In order to avoid user confusion, as a matter of policy all GUI tools display the word *DFHack* on the screen somewhere while active.

When that is not appropriate because they merely add keybinding hints to existing DF screens, they deliberately use red instead of green for the key.

---

automelt

When automelt is enabled for a stockpile, any meltable items placed in it will be designated to be melted. This plugin adds an option to the `q` menu when *enabled*.

autotrade

When autotrade is enabled for a stockpile, any items placed in it will be designated to be taken to the Trade Depot whenever merchants are on the map. This plugin adds an option to the `q` menu when *enabled*.

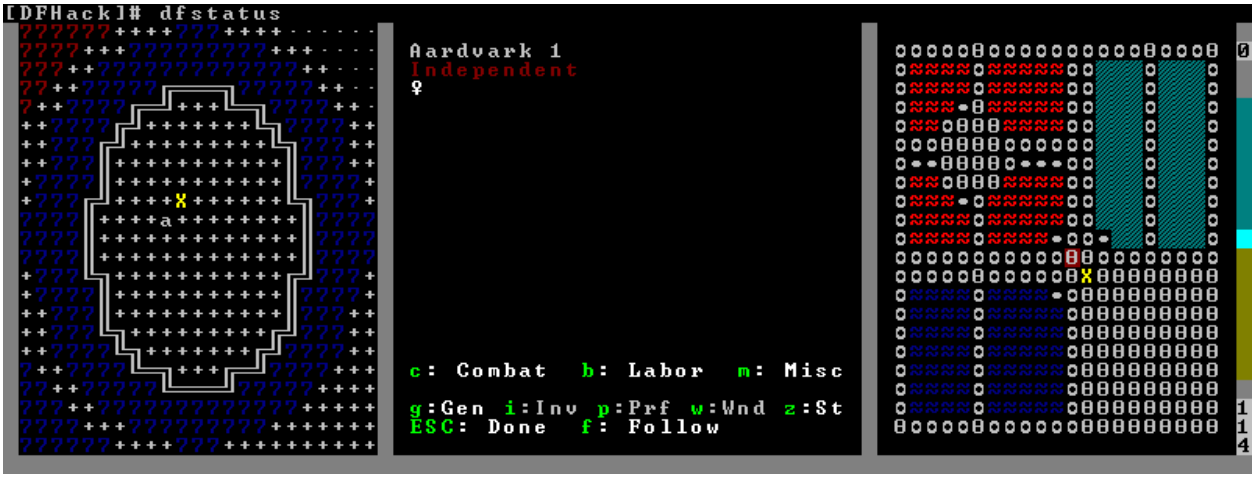
command-prompt

An in-game DFHack terminal, where you can enter other commands. Best used from a keybinding; by default `CtrlShiftP`.

Usage: `command-prompt [entry]`

If called with an entry, it starts with that text filled in. Most useful for developers, who can set a keybinding to open a laungage interpreter for lua or Ruby by starting with the *:lua* or *:rb* commands.

Otherwise somewhat similar to *gui/quickcmd*.



hotkeys

Opens an in-game screen showing which DFHack keybindings are active in the current context.

Type `hotkeys` into the DFHack console to open the screen, or bind the command to a globally active hotkey. The default keybinding is `CtrlF1`. See also *hotkey-notes*.

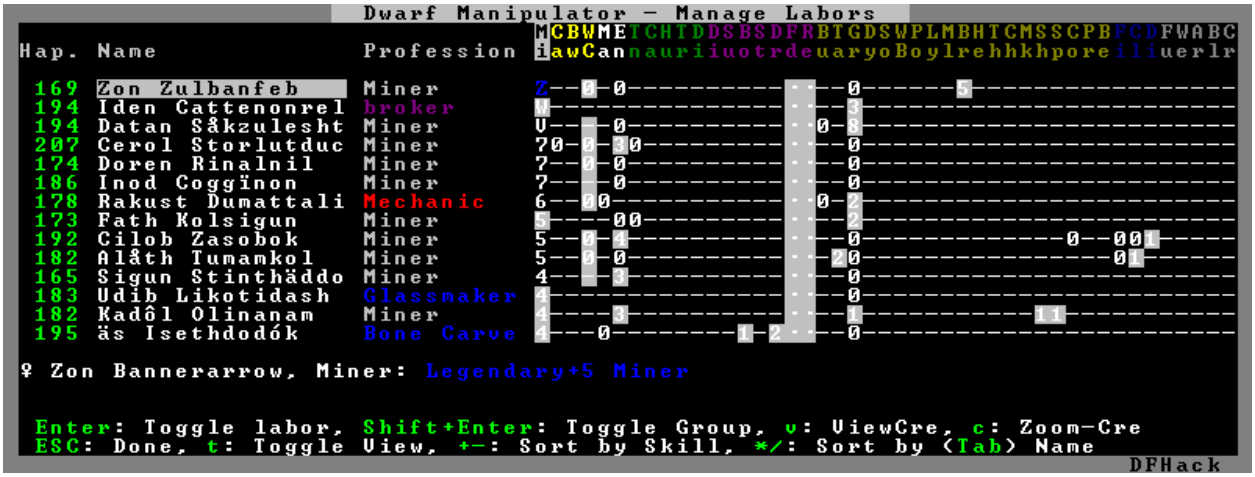
rb

Ruby language plugin, which evaluates the following arguments as a ruby string. Best used as `:rb [string]`, for the special parsing mode. Alias `rb_eval`.

manipulator

An in-game equivalent to the popular program Dwarf Therapist.

To activate, open the unit screen and press `l`.



The far left column displays the unit's Happiness (color-coded based on its value), Name, Profession/Squad, and the right half of the screen displays each dwarf's labor settings and skill levels (0-9 for Dabbling through Professional, A-E for Great through Grand Master, and U-Z for Legendary through Legendary+5).

Cells with teal backgrounds denote skills not controlled by labors, e.g. military and social skills.

Press `t` to toggle between Profession and Squad view.

Use the arrow keys or number pad to move the cursor around, holding `Shift` to move 10 tiles at a time.

Press the Z-Up (<) and Z-Down (>) keys to move quickly between labor/skill categories. The numpad Z-Up and

Z-Down keys seek to the first or last unit in the list. `Backspace` seeks to the top left corner.

Press Enter to toggle the selected labor for the selected unit, or Shift+Enter to toggle all labors within the selected category.

Press the `+-` keys to sort the unit list according to the currently selected skill/labor, and press the `*/` keys to sort the unit list by Name, Profession/Squad, Happiness, or Arrival order (using `Tab` to select which sort method to use here).

With a unit selected, you can press the `v` key to view its properties (and possibly set a custom nickname or profession) or the `c` key to exit Manipulator and zoom to its position within your fortress.

The following mouse shortcuts are also available:

- Click on a column header to sort the unit list. Left-click to sort it in one direction (descending for happiness or labors/skills, ascending for name, profession or squad) and right-click to sort it in the opposite direction.
- Left-click on a labor cell to toggle that labor. Right-click to move the cursor onto that cell instead of toggling it.
- Left-click on a unit's name, profession or squad to view its properties.
- Right-click on a unit's name, profession or squad to zoom to it.

Pressing `Esc` normally returns to the unit screen, but `ShiftEsc` would exit directly to the main dwarf mode screen.

search

The search plugin adds search to the Stocks, Animals, Trading, Stockpile, Noble (assignment candidates), Military (position candidates), Burrows (unit list), Rooms, Announcements, Job List and Unit List screens.

Searching works the same way as the search option in *Move to Depot*. You will see the Search option displayed on screen with a hotkey (usually `s`). Pressing it lets you start typing a query and the relevant list will start filtering automatically.

Pressing `Enter`, `Esc` or the arrow keys will return you to browsing the now filtered list, which still functions as normal. You can clear the filter by either going back into search mode and backspacing to delete it, or pressing the "shifted" version of the search hotkey while browsing the list (e.g. if the hotkey is `s`, then hitting `Shifts` will clear any filter).

Leaving any screen automatically clears the filter.

In the Trade screen, the actual trade will always only act on items that are actually visible in the list; the same effect applies to the Trade Value numbers displayed by the screen. Because of this, the `t` key is blocked while search is active, so you have to reset the filters first. Pressing `AltC` will clear both search strings.

In the stockpile screen the option only appears if the cursor is in the rightmost list:

Note that the 'Permit XXX'/'Forbid XXX' keys conveniently operate only on items actually shown in the rightmost list, so it is possible to select only fat or tallow by forbidding fats, then searching for fat/tallow, and using Permit Fats again while the list is filtered.

nopause

Disables pausing (both manual and automatic) with the exception of pause forced by *reveal* `hell`. This is nice for digging under rivers.

embark-tools

A collection of embark-related tools. Usage and available tools:

```
embark-tools enable/disable tool [tool]...
```

anywhere Allows embarking anywhere (including sites, mountain-only biomes, and oceans). Use with caution.

mouse Implements mouse controls (currently in the local embark region only)

sand Displays an indicator when sand is present in the currently-selected area, similar to the default clay/stone indicators.

sticky Maintains the selected local area while navigating the world map

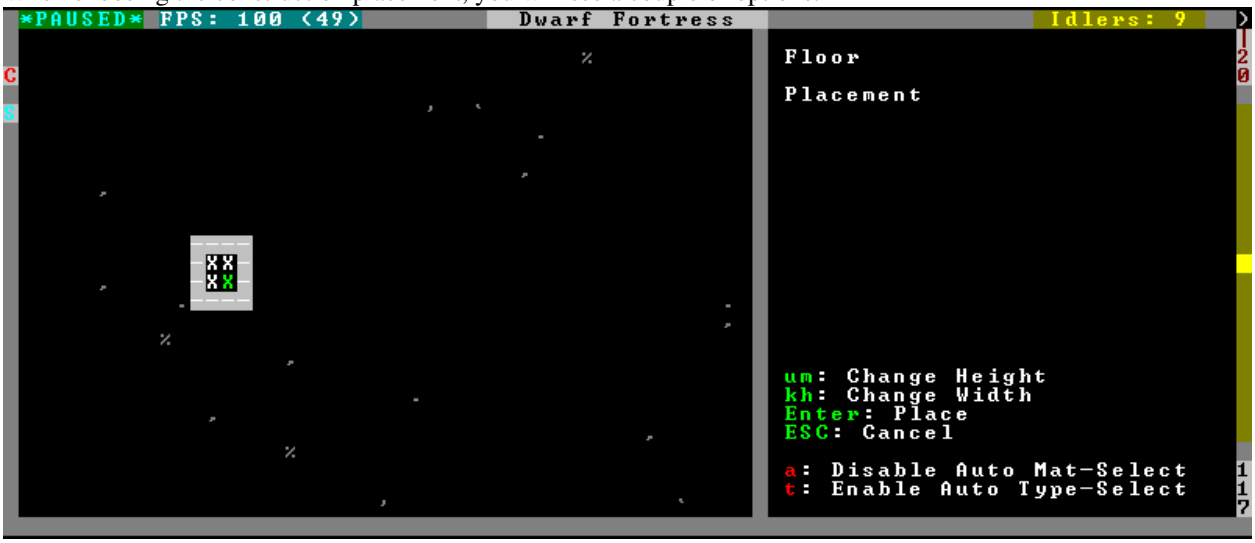
automaterial

This makes building constructions (walls, floors, fortifications, etc) a little bit easier by saving you from having to trawl through long lists of materials each time you place one.

Firstly, it moves the last used material for a given construction type to the top of the list, if there are any left. So if you build a wall with chalk blocks, the next time you place a wall the chalk blocks will be at the top of the list, regardless of distance (it only does this in "grouped" mode, as individual item lists could be huge). This should mean you can place most constructions without having to search for your preferred material type.

Pressing `a` while highlighting any material will enable that material for "auto select" for this construction type. You can enable multiple materials as autoselect. Now the next time you place this type of construction, the plugin will automatically choose materials for you from the kinds you enabled. If there is enough to satisfy the whole placement, you won't be prompted with the material screen - the construction will be placed and you will be back in the construction menu as if you did it manually.

When choosing the construction placement, you will see a couple of options:



Use `a` here to temporarily disable the material autoselection, e.g. if you need to go to the material selection screen so you can toggle some materials on or off.

The other option (auto type selection, off by default) can be toggled on with `t`. If you toggle this option on, instead of returning you to the main construction menu after selecting materials, it returns you back to this screen. If you use this along with several autoselect enabled materials, you should be able to place complex constructions more conveniently.

buildingplan

When active (via `enable buildingplan`), this plugin adds a planning mode for furniture placement. You can then place furniture and other buildings before the required materials are available, and the job will be unsuspended when the item is created.

Very useful when combined with *workflow* - you can set a constraint to always have one or two doors/beds/tables/chairs/etc available, and place as many as you like. The plugins then take over and fulfill the orders, with minimal space dedicated to stockpiles.

confirm

Implements several confirmation dialogs for potentially destructive actions (for example, seizing goods from traders or deleting hauling routes).

Usage:

enable confirm Enable all confirmations; alias `confirm enable all`. Replace with `disable` to disable.

confirm help List available confirmation dialogues.

confirm enable option1 [option2...] Enable (or disable) specific confirmation dialogues.

follow

Makes the game view follow the currently highlighted unit after you exit from the current menu or cursor mode. Handy for watching dwarves running around. Deactivated by moving the view manually.

mousequery

Adds mouse controls to the DF interface, eg click-and-drag designations.

Options:

plugin enable/disable the entire plugin

rbutton enable/disable right mouse button

track enable/disable moving cursor in build and designation mode

edge enable/disable active edge scrolling (when on, will also enable tracking)

live enable/disable query view when unpaused

delay Set delay when edge scrolling in tracking mode. Omit amount to display current setting.

Usage:

```
mousequery [plugin] [rbutton] [track] [edge] [live] [enable|disable]
```

resume

Allows automatic resumption of suspended constructions, along with colored UI hints for construction status.

title-version

Displays the DFHack version on DF's title screen when enabled.

trackstop

Adds a `q` menu for track stops, which is completely blank by default. This allows you to view and/or change the track stop's friction and dump direction settings, using the keybindings from the track stop building interface.

sort-items

Sort the visible item list:

```
sort-items order [order...]
```

Sort the item list using the given sequence of comparisons. The < prefix for an order makes undefined values sort first. The > prefix reverses the sort order for defined values.

Item order examples:

```
description material wear type quality
```

The orderings are defined in `hack/lua/plugins/sort/*.lua`

sort-units

Sort the visible unit list:

```
sort-units order [order...]
```

Sort the unit list using the given sequence of comparisons. The < prefix for an order makes undefined values sort first. The > prefix reverses the sort order for defined values.

Unit order examples:

```
name age arrival squad squad_position profession
```

The orderings are defined in `hack/lua/plugins/sort/*.lua`

stocks

Replaces the DF stocks screen with an improved version.

stocksettings

Offers the following commands to save and load stockpile settings. See *gui/stockpiles* for an in-game interface.

copystock Copies the parameters of the currently highlighted stockpile to the custom stockpile settings and switches to custom stockpile placement mode, effectively allowing you to copy/paste stockpiles easily.

savestock Saves the currently highlighted stockpile's settings to a file in your Dwarf Fortress folder. This file can be used to copy settings between game saves or players. eg: `savestock food_settings.dfstock`

loadstock Loads a saved stockpile settings file and applies it to the currently selected stockpile. eg: `loadstock food_settings.dfstock`

To use savestock and loadstock, use the `q` command to highlight a stockpile. Then run savestock giving it a descriptive filename. Then, in a different (or the same!) gameworld, you can highlight any stockpile with `q` then execute the `loadstock` command passing it the name of that file. The settings will be applied to that stockpile.

Note that files are relative to the DF folder, so put your files there or in a subfolder for easy access. Filenames should not have spaces. Generated materials, divine metals, etc are not saved as they are different in every world.

rename

Allows renaming various things. Use *gui/rename* for an in-game interface.

Options:

`rename squad <index> "name"` Rename squad by index to 'name'.

`rename hotkey <index> \"name\"` Rename hotkey by index. This allows assigning longer commands to the DF hotkeys.

`rename unit "nickname"` Rename a unit/creature highlighted in the DF user interface.

`rename unit-profession "custom profession"` Change proffession name of the highlighted unit/creature.

`rename building "name"` Set a custom name for the selected building. The building must be one of stockpile, workshop, furnace, trap, siege engine or an activity zone.

rendermax

A collection of renderer replacing/enhancing filters. For better effect try changing the black color in palette to non totally black. See Bay12 forums thread 128487 for more info.

Options:

trippy Randomizes the color of each tiles. Used for fun, or testing.

light Enable lighting engine.

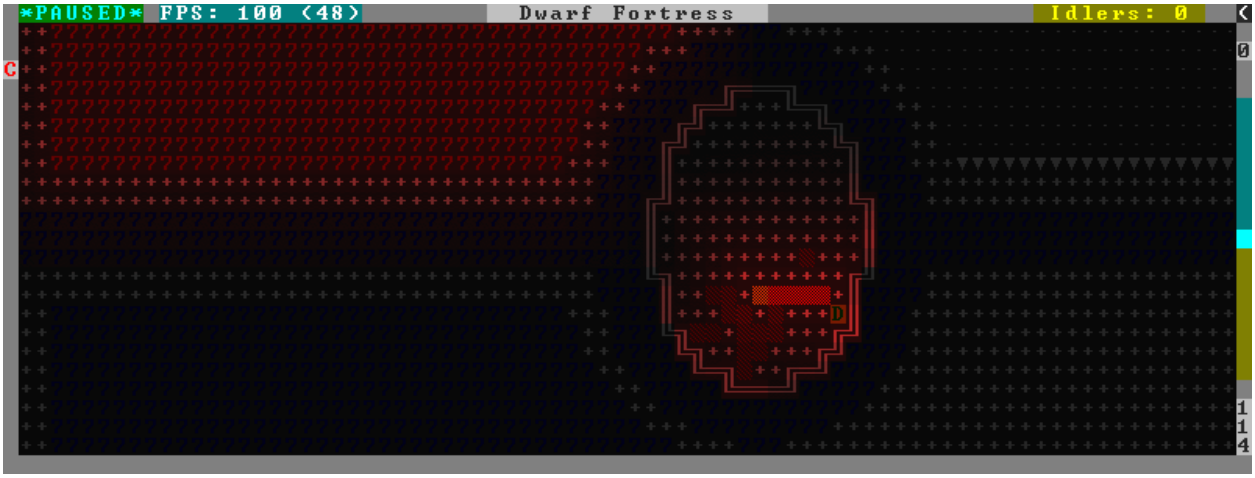
light reload Reload the settings file.

light sun <x>|cycle Set time to <x> (in hours) or set it to df time cycle.

occlusionON, occlusionOFF Show debug occlusion info.

disable Disable any filter that is enabled.

An image showing lava and dragon breath. Not pictured here: sunlight, shining items/plants, materials that color the light etc...



2.3.4 Job and Fortress management

autolabor

Automatically manage dwarf labors to efficiently complete jobs. Autolabor tries to keep as many dwarves as possible busy but also tries to have dwarves specialize in specific skills.

The key is that, for almost all labors, once a dwarf begins a job it will finish that job even if the associated labor is removed. Autolabor therefore frequently checks which dwarf or dwarves should take new jobs for that labor, and sets labors accordingly. Labors with equiptment (mining, hunting, and woodcutting), which are abandoned if labors change mid-job, are handled slightly differently to minimise churn.

Warning: *autolabor will override any manual changes you make to labors while it is enabled, including through other tools such as Dwarf Therapist*

Simple usage:

enable autolabor Enables the plugin with default settings. (Persistent per fortress)

disable autolabor Disables the plugin.

Anything beyond this is optional - autolabor works well on the default settings.

By default, each labor is assigned to between 1 and 200 dwarves (2-200 for mining). By default 33% of the workforce become haulers, who handle all hauling jobs as well as cleaning, pulling levers, recovering wounded, removing constructions, and filling ponds. Other jobs are automatically assigned as described above. Each of these settings can be adjusted.

Jobs are rarely assigned to nobles with responsibilities for meeting diplomats or merchants, never to the chief medical dwarf, and less often to the bookeeper and manager.

Hunting is never assigned without a butchery, and fishing is never assigned without a fishery.

For each labor a preference order is calculated based on skill, biased against masters of other trades and excluding those who can't do the job. The labor is then added to the best <minimum> dwarves for that labor. We assign at least the minimum number of dwarfs, in order of preference, and then assign additional dwarfs that meet any of these conditions:

- The dwarf is idle and there are no idle dwarves assigned to this labor
- The dwarf has non-zero skill associated with the labor
- The labor is mining, hunting, or woodcutting and the dwarf currently has it enabled.

We stop assigning dwarfs when we reach the maximum allowed.

Advanced usage:

autolabor <labor> <minimum> [<maximum>] Set number of dwarves assigned to a labor.

autolabor <labor> haulers Set a labor to be handled by hauler dwarves.

autolabor <labor> disable Turn off autolabor for a specific labor.

autolabor <labor> reset Return a labor to the default handling.

autolabor reset-all Return all labors to the default handling.

autolabor list List current status of all labors.

autolabor status Show basic status information.

See *autolabor-artisans* for a differently-tuned setup.

Examples:

`autolabor MINE` Keep at least 5 dwarves with mining enabled.

`autolabor CUT_GEM 1 1` Keep exactly 1 dwarf with gemcutting enabled.

`autolabor COOK 1 1 3` Keep 1 dwarf with cooking enabled, selected only from the top 3.

`autolabor FEED_WATER_CIVILIANS haulers` Have haulers feed and water wounded dwarves.

`autolabor CUTWOOD disable` Turn off autolabor for wood cutting.

autohauler

Autohauler is an autolabor fork.

Rather than the all-of-the-above means of autolabor, autohauler will instead only manage hauling labors and leave skilled labors entirely to the user, who will probably use Dwarf Therapist to do so.

Idle dwarves will be assigned the hauling labors; everyone else (including those currently hauling) will have the hauling labors removed. This is to encourage every dwarf to do their assigned skilled labors whenever possible, but resort to hauling when those jobs are not available. This also implies that the user will have a very tight skill assignment, with most skilled labors only being assigned to just one dwarf, no dwarf having more than two active skilled labors, and almost every non-military dwarf having at least one skilled labor assigned.

Autohauler allows skills to be flagged as to prevent hauling labors from being assigned when the skill is present. By default this is the unused ALCHEMIST labor but can be changed by the user.

job

Command for general job query and manipulation.

Options:

no extra options Print details of the current job. The job can be selected in a workshop, or the unit/jobs screen.

list Print details of all jobs in the selected workshop.

item-material <item-idx> <material[:subtoken]> Replace the exact material id in the job item.

item-type <item-idx> <type[:subtype]> Replace the exact item type id in the job item.

job-material

Alter the material of the selected job. Similar to `job item-material ...`

Invoked as:

```
job-material <inorganic-token>
```

Intended to be used as a keybinding:

- In `q` mode, when a job is highlighted within a workshop or furnace, changes the material of the job. Only inorganic materials can be used in this mode.
- In `b` mode, during selection of building components positions the cursor over the first available choice with the matching material.

job-duplicate

In `q` mode, when a job is highlighted within a workshop or furnace building, calling `job-duplicate` instantly duplicates the job.

autogems

Creates a new Workshop Order setting, automatically cutting rough gems when *enabled*.

stockflow

Allows the fortress bookkeeper to queue jobs through the manager, based on space or items available in stockpiles.

Inspired by *workflow*.

Usage:

`stockflow enable` Enable the plugin.

`stockflow disable` Disable the plugin.

`stockflow fast` Enable the plugin in fast mode.

`stockflow list` List any work order settings for your stockpiles.

`stockflow status` Display whether the plugin is enabled.

While enabled, the `q` menu of each stockpile will have two new options:

- `j`: Select a job to order, from an interface like the manager's screen.
- `J`: Cycle between several options for how many such jobs to order.

Whenever the bookkeeper updates stockpile records, new work orders will be placed on the manager's queue for each such selection, reduced by the number of identical orders already in the queue.

In fast mode, new work orders will be enqueued once per day, instead of waiting for the bookkeeper.

workflow

Manage control of repeat jobs. *gui/workflow* provides a simple front-end integrated in the game UI.

Usage:

`workflow enable [option...], workflow disable [option...]` If no options are specified, enables or disables the plugin. Otherwise, enables or disables any of the following options:

- drybuckets: Automatically empty abandoned water buckets.
- auto-melt: Resume melt jobs when there are objects to melt.

`workflow jobs` List workflow-controlled jobs (if in a workshop, filtered by it).

`workflow list` List active constraints, and their job counts.

`workflow list-commands` List active constraints as workflow commands that re-create them; this list can be copied to a file, and then reloaded using the `script` built-in command.

`workflow count <constraint-spec> <cnt-limit> [cnt-gap]` Set a constraint, counting every stack as 1 item.

`workflow amount <constraint-spec> <cnt-limit> [cnt-gap]` Set a constraint, counting all items within stacks.

`workflow unlimit <constraint-spec>` Delete a constraint.

`workflow unlimit-all` Delete all constraints.

Function

When the plugin is enabled, it protects all repeat jobs from removal. If they do disappear due to any cause, they are immediately re-added to their workshop and suspended.

In addition, when any constraints on item amounts are set, repeat jobs that produce that kind of item are automatically suspended and resumed as the item amount goes above or below the limit. The gap specifies how much below the limit the amount has to drop before jobs are resumed; this is intended to reduce the frequency of jobs being toggled.

Constraint format

The constraint spec consists of 4 parts, separated with / characters:

```
ITEM[:SUBTYPE]/[GENERIC_MAT,...]/[SPECIFIC_MAT:...]/[LOCAL,<quality>]
```

The first part is mandatory and specifies the item type and subtype, using the raw tokens for items (the same syntax used custom reaction inputs). For more information, see this wiki page.

The subsequent parts are optional:

- A generic material spec constrains the item material to one of the hard-coded generic classes, which currently include:

```
    PLANT WOOD CLOTH SILK LEATHER BONE SHELL SOAP TOOTH HORN PEARL YARN
    METAL STONE SAND GLASS CLAY MILK
```

- A specific material spec chooses the material exactly, using the raw syntax for reaction input materials, e.g. `INORGANIC:IRON`, although for convenience it also allows just `IRON`, or `ACACIA:WOOD` etc. See the link above for more details on the unabbreviated raw syntax.
- A comma-separated list of miscellaneous flags, which currently can be used to ignore imported items or items below a certain quality.

Constraint examples

Keep metal bolts within 900-1000, and wood/bone within 150-200:

```
workflow amount AMMO:ITEM_AMMO_BOLTS/METAL 1000 100
workflow amount AMMO:ITEM_AMMO_BOLTS/WOOD,BONE 200 50
```

Keep the number of prepared food & drink stacks between 90 and 120:

```
workflow count FOOD 120 30
workflow count DRINK 120 30
```

Make sure there are always 25-30 empty bins/barrels/bags:

```
workflow count BIN 30
workflow count BARREL 30
workflow count BOX/CLOTH,SILK,YARN 30
```

Make sure there are always 15-20 coal and 25-30 copper bars:

```
workflow count BAR//COAL 20
workflow count BAR//COPPER 30
```

Produce 15-20 gold crafts:

```
workflow count CRAFTS//GOLD 20
```

Collect 15-20 sand bags and clay boulders:

```
workflow count POWDER_MISC/SAND 20
workflow count BOULDER/CLAY 20
```

Make sure there are always 80-100 units of dimple dye:

```
workflow amount POWDER_MISC//MUSHROOM_CUP_DIMPLE:MILL 100 20
```

Note: In order for this to work, you have to set the material of the PLANT input on the Mill Plants job to MUSHROOM_CUP_DIMPLE using the *job item-material* command. Otherwise the plugin won't be able to deduce the output material.

Maintain 10-100 locally-made crafts of exceptional quality:

```
workflow count CRAFTS///LOCAL,EXCEPTIONAL 100 90
```

fix-job-postings

This command fixes crashes caused by previous versions of workflow, mostly in DFHack 0.40.24-r4, and should be run automatically when loading a world (but can also be run manually if desired).

clean

Cleans all the splatter that get scattered all over the map, items and creatures. In an old fortress, this can significantly reduce FPS lag. It can also spoil your !!FUN!!, so think before you use it.

Options:

map Clean the map tiles. By default, it leaves mud and snow alone.

units Clean the creatures. Will also clean hostiles.

items Clean all the items. Even a poisoned blade.

Extra options for `map`:

mud Remove mud in addition to the normal stuff.

snow Also remove snow coverings.

spotclean

Works like `clean map snow mud`, but only for the tile under the cursor. Ideal if you want to keep that bloody entrance `clean map` would clean up.

autodump

This plugin adds an option to the `q` menu for stckpiles when *enabled*. When autodump is enabled for a stockpile, any items placed in the stockpile will automatically be designated to be dumped.

Alternatively, you can use it to quickly move all items designated to be dumped. Items are instantly moved to the cursor position, the dump flag is unset, and the forbid flag is set, as if it had been dumped normally. Be aware that any active dump item tasks still point at the item.

Cursor must be placed on a floor tile so the items can be dumped there.

Options:

destroy Destroy instead of dumping. Doesn't require a cursor. If called again before the game is resumed, cancels destroy.

destroy-here As `destroy`, but only the selected item in the `k` list, or inside a container. Alias `autodump-destroy-here`, for keybindings.

visible Only process items that are not hidden.

hidden Only process hidden items.

forbidden Only process forbidden items (default: only unforbidden).

cleanowned

Confiscates items owned by dwarfs. By default, owned food on the floor and rotten items are confistacted and dumped.

Options:

all confiscate all owned items

scattered confiscated and dump all items scattered on the floor

x confiscate/dump items with wear level 'x' and more

X confiscate/dump items with wear level 'X' and more

dryrun a dry run. combine with other options to see what will happen without it actually happening.

Example:

`cleanowned scattered X` This will confiscate rotten and dropped food, garbage on the floors and any worn items with 'X' damage and above.

dwarfmonitor

Records dwarf activity to measure fort efficiency.

Options:

enable <mode> Start monitoring `mode`. `mode` can be "work", "misery", "weather", or "all". This will enable all corresponding widgets, if applicable.

disable <mode> Stop monitoring `mode`, and disable corresponding widgets, if applicable.

stats Show statistics summary

prefs Show dwarf preferences summary

reload Reload configuration file (`dfhack-config/dwarfmonitor.json`)

Widget configuration:

The following types of widgets (defined in `hack/lua/plugins/dwarfmonitor.lua`) can be displayed on the main fortress mode screen:

date Show the in-game date

misery Show overall happiness levels of all dwarves

weather Show current weather (rain/snow)

cursor Show the current mouse cursor position

The file `dfhack-config/dwarfmonitor.json` can be edited to control the positions and settings of all widgets displayed. This file should contain a JSON object with the key `widgets` containing an array of objects - see the included file in the `dfhack-config` folder for an example:

```
{
    "widgets": [
        {
            "type": "widget type (weather, misery, etc.)",
            "x": X coordinate,
            "y": Y coordinate
            <...additional options...>
        }
    ]
}
```

X and Y coordinates begin at zero (in the upper left corner of the screen). Negative coordinates will be treated as distances from the lower right corner, beginning at 1 - e.g. an x coordinate of 0 is the leftmost column, while an x coordinate of 1 is the rightmost column.

By default, the x and y coordinates given correspond to the leftmost tile of the widget. Including an `anchor` option set to `right` will cause the rightmost tile of the widget to be located at this position instead.

Some widgets support additional options:

- `date` widget:
  - `format`: specifies the format of the date. The following characters are replaced (all others, such as punctuation, are not modified)
    - `Y` or `y`: The current year
    - `M`: The current month, zero-padded if necessary
    - `m`: The current month, *not* zero-padded
    - `D`: The current day, zero-padded if necessary
    - `d`: The current day, *not* zero-padded

    The default date format is `Y-M-D`, per the ISO8601 standard.
- `cursor` widget:
  - `format`: Specifies the format. `X`, `x`, `Y`, and `y` are replaced with the corresponding cursor cordinates, while all other characters are unmodified.
  - `show_invalid`: If set to `true`, the mouse coordinates will both be displayed as `-1` when the cursor is outside of the DF window; otherwise, nothing will be displayed.

workNow

Force all dwarves to look for a job immediately, or as soon as the game is unpaused.

seedwatch

Watches the numbers of seeds available and enables/disables seed and plant cooking.

Each plant type can be assigned a limit. If their number falls below that limit, the plants and seeds of that type will be excluded from cookery. If the number rises above the limit + 20, then cooking will be allowed.

The plugin needs a fortress to be loaded and will deactivate automatically otherwise. You have to reactivate with 'seedwatch start' after you load the game.

Options:

all Adds all plants from the abbreviation list to the watch list.

start Start watching.

stop Stop watching.

info Display whether seedwatch is watching, and the watch list.

clear Clears the watch list.

Examples:

`seedwatch MUSHROOM_HELMET_PLUMP 30` add `MUSHROOM_HELMET_PLUMP` to the watch list, limit = 30

`seedwatch MUSHROOM_HELMET_PLUMP` removes `MUSHROOM_HELMET_PLUMP` from the watch list.

`seedwatch all 30` adds all plants from the abbreviation list to the watch list, the limit being 30.

zone

Helps a bit with managing activity zones (pens, pastures and pits) and cages.

Options:

set Set zone or cage under cursor as default for future assigns.

assign Assign unit(s) to the pen or pit marked with the 'set' command. If no filters are set a unit must be selected in the in-game ui. Can also be followed by a valid zone id which will be set instead.

unassign Unassign selected creature from it's zone.

nick Mass-assign nicknames, must be followed by the name you want to set.

remnick Mass-remove nicknames.

tocages Assign unit(s) to cages inside a pasture.

uinfo Print info about unit(s). If no filters are set a unit must be selected in the in-game ui.

zinfo Print info about zone(s). If no filters are set zones under the cursor are listed.

verbose Print some more info.

filters Print list of valid filter options.

examples Print some usage examples.

not Negates the next filter keyword.

Filters:

all Process all units (to be used with additional filters).

count Must be followed by a number. Process only n units (to be used with additional filters).

unassigned Not assigned to zone, chain or built cage.

minage Minimum age. Must be followed by number.

maxage Maximum age. Must be followed by number.

race Must be followed by a race RAW ID (e.g. BIRD_TURKEY, ALPACA, etc). Negatable.

caged In a built cage. Negatable.

own From own civilization. Negatable.

merchant Is a merchant / belongs to a merchant. Should only be used for pitting, not for stealing animals (slaughter should work).

war Trained war creature. Negatable.

hunting Trained hunting creature. Negatable.

tamed Creature is tame. Negatable.

trained Creature is trained. Finds war/hunting creatures as well as creatures who have a training level greater than 'domesticated'. If you want to specifically search for war/hunting creatures use 'war' or 'hunting' Negatable.

trainablewar Creature can be trained for war (and is not already trained for war/hunt). Negatable.

trainablehunt Creature can be trained for hunting (and is not already trained for war/hunt). Negatable.

male Creature is male. Negatable.

female Creature is female. Negatable.

egglayer Race lays eggs. Negatable.

grazer Race is a grazer. Negatable.

milkable Race is milkable. Negatable.

Usage with single units

One convenient way to use the zone tool is to bind the command 'zone assign' to a hotkey, maybe also the command 'zone set'. Place the in-game cursor over a pen/pasture or pit, use 'zone set' to mark it. Then you can select units on the map (in 'v' or 'k' mode), in the unit list or from inside cages and use 'zone assign' to assign them to their new home. Allows pitting your own dwarves, by the way.

Usage with filters

All filters can be used together with the 'assign' command.

Restrictions: It's not possible to assign units who are inside built cages or chained because in most cases that won't be desirable anyways. It's not possible to cage owned pets because in that case the owner uncages them after a while which results in infinite hauling back and forth.

Usually you should always use the filter 'own' (which implies tame) unless you want to use the zone tool for pitting hostiles. 'own' ignores own dwarves unless you specify 'race DWARF' (so it's safe to use 'assign all own' to one big pasture if you want to have all your animals at the same place). 'egglayer' and 'milkable' should be used together with 'female' unless you have a mod with egg-laying male elves who give milk or whatever. Merchants and their animals are ignored unless you specify 'merchant' (pitting them should be no problem, but stealing and pasturing their animals is not a good idea since currently they are not properly added to your own stocks; slaughtering them should work).

Most filters can be negated (e.g. 'not grazer' -> race is not a grazer).

Mass-renaming

Using the 'nick' command you can set the same nickname for multiple units. If used without 'assign', 'all' or 'count' it will rename all units in the current default target zone. Combined with 'assign', 'all' or 'count' (and further optional filters) it will rename units matching the filter conditions.

Cage zones

Using the 'tocages' command you can assign units to a set of cages, for example a room next to your butcher shop(s). They will be spread evenly among available cages to optimize hauling to and butchering from them. For this to work

you need to build cages and then place one pen/pasture activity zone above them, covering all cages you want to use. Then use 'zone set' (like with 'assign') and use 'zone tocages filter1 filter2 ...'. 'tocages' overwrites 'assign' because it would make no sense, but can be used together with 'nick' or 'remnick' and all the usual filters.

Examples

`zone assign all own ALPACA minage 3 maxage 10` Assign all own alpacas who are between 3 and 10 years old to the selected pasture.

`zone assign all own caged grazer nick ineedgrass` Assign all own grazers who are sitting in cages on stockpiles (e.g. after buying them from merchants) to the selected pasture and give them the nickname 'ineedgrass'.

`zone assign all own not grazer not race CAT` Assign all own animals who are not grazers, excluding cats.

`zone assign count 5 own female milkable` Assign up to 5 own female milkable creatures to the selected pasture.

`zone assign all own race DWARF maxage 2` Throw all useless kids into a pit :)

`zone nick donttouchme` Nicknames all units in the current default zone or cage to 'donttouchme'. Mostly intended to be used for special pastures or cages which are not marked as rooms you want to protect from autobutcher.

`zone tocages count 50 own tame male not grazer` Stuff up to 50 owned tame male animals who are not grazers into cages built on the current default zone.

autonestbox

Assigns unpastured female egg-layers to nestbox zones. Requires that you create pen/pasture zones above nestboxes. If the pen is bigger than 1x1 the nestbox must be in the top left corner. Only 1 unit will be assigned per pen, regardless of the size. The age of the units is currently not checked, most birds grow up quite fast. Egglayers who are also grazers will be ignored, since confining them to a 1x1 pasture is not a good idea. Only tame and domesticated own units are processed since pasturing half-trained wild egglayers could destroy your neat nestbox zones when they revert to wild. When called without options autonestbox will instantly run once.

Options:

start Start running every X frames (df simulation ticks). Default: X=6000, which would be every 60 seconds at 100fps.

stop Stop running automatically.

sleep Must be followed by number X. Changes the timer to sleep X frames between runs.

autobutcher

Assigns lifestock for slaughter once it reaches a specific count. Requires that you add the target race(s) to a watch list. Only tame units will be processed.

Units will be ignored if they are:

- Nicknamed (for custom protection; you can use the *rename* `unit` tool individually, or *zone* `nick` for groups)
- Caged, if and only if the cage is defined as a room (to protect zoos)
- Trained for war or hunting

Creatures who will not reproduce (because they're not interested in the opposite sex or have been gelded) will be butchered before those who will. Older adults and younger children will be butchered first if the population is above the target (default 1 male, 5 female kids and adults). Note that you may need to set a target above 1 to have a reliable breeding population due to asexuality etc. See *fix-ster* if this is a problem.

Options:

example Print some usage examples.

start Start running every X frames (df simulation ticks). Default: X=6000, which would be every 60 seconds at 100fps.

stop Stop running automatically.

sleep <x> Changes the timer to sleep X frames between runs.

watch R Start watching a race. R can be a valid race RAW id (ALPACA, BIRD_TURKEY, etc) or a list of ids seperated by spaces or the keyword 'all' which affects all races on your current watchlist.

unwatch R Stop watching race(s). The current target settings will be remembered. R can be a list of ids or the keyword 'all'.

forget R Stop watching race(s) and forget it's/their target settings. R can be a list of ids or the keyword 'all'.

autowatch Automatically adds all new races (animals you buy from merchants, tame yourself or get from migrants) to the watch list using default target count.

noautowatch Stop auto-adding new races to the watchlist.

list Print the current status and watchlist.

list_export Print the commands needed to set up status and watchlist, which can be used to import them to another save (see notes).

target <fk> <mk> <fa> <ma> <R> Set target count for specified race(s). The first four arguments are the number of female and male kids, and female and male adults. R can be a list of spceies ids, or the keyword `all` or `new`. `R = 'all'`: change target count for all races on watchlist and set the new default for the future. `R = 'new'`: don't touch current settings on the watchlist, only set the new default for future entries.

list_export Print the commands required to rebuild your current settings.

Note: Settings and watchlist are stored in the savegame, so that you can have different settings for each save. If you want to copy your watchlist to another savegame you must export the commands required to recreate your settings.

To export, open an external terminal in the DF directory, and run `dfhack-run autobutcher list_export > filename.txt`. To import, load your new save and run `script filename.txt` in the DFHack terminal.

Examples:

You want to keep max 7 kids (4 female, 3 male) and max 3 adults (2 female, 1 male) of the race alpaca. Once the kids grow up the oldest adults will get slaughtered. Excess kids will get slaughtered starting with the youngest to allow that the older ones grow into adults. Any unnamed cats will be slaughtered as soon as possible.

```
autobutcher target 4 3 2 1 ALPACA BIRD_TURKEY
autobutcher target 0 0 0 0 CAT
autobutcher watch ALPACA BIRD_TURKEY CAT
autobutcher start
```

Automatically put all new races onto the watchlist and mark unnamed tame units for slaughter as soon as they arrive in your fort. Settings already made for specific races will be left untouched.

```
autobutcher target 0 0 0 0 new
autobutcher autowatch
autobutcher start
```

Stop watching the races alpaca and cat, but remember the target count settings so that you can use 'unwatch' without the need to enter the values again. Note: 'autobutcher unwatch all' works, but only makes sense if you want to keep the plugin running with the 'autowatch' feature or manually add some new races with 'watch'. If you simply want to stop it completely use 'autobutcher stop' instead.

```
autobutcher unwatch ALPACA CAT
```

autochop

Automatically manage tree cutting designation to keep available logs withing given quotas.

Open the dashboard by running:

```
getplants autochop
```

The plugin must be activated (with `c`) before it can be used. You can then set logging quotas and restrict designations to specific burrows (with 'Enter') if desired. The plugin's activity cycle runs once every in game day.

If you add `enable getplants` to your dfhack.init there will be a hotkey to open the dashboard from the chop designation menu.

2.3.5 Map modification

3dveins

Removes all existing veins from the map and generates new ones using 3D Perlin noise, in order to produce a layout that smoothly flows between Z levels. The vein distribution is based on the world seed, so running the command for the second time should produce no change. It is best to run it just once immediately after embark.

This command is intended as only a cosmetic change, so it takes care to exactly preserve the mineral counts reported by *prospect* `all`. The amounts of different layer stones may slightly change in some cases if vein mass shifts between Z layers.

The only undo option is to restore your save from backup.

alltraffic

Set traffic designations for every single tile of the map - useful for resetting traffic designations. See also *filltraffic*, *restrictice*, and *restrictliquids*.

Options:

H High Traffic

N Normal Traffic

L Low Traffic

R Restricted Traffic

burrow

Miscellaneous burrow control. Allows manipulating burrows and automated burrow expansion while digging.

Options:

enable feature ... Enable features of the plugin.

disable feature ... Disable features of the plugin.

clear-unit burrow burrow ... Remove all units from the burrows.

clear-tiles burrow burrow ... Remove all tiles from the burrows.

set-units target-burrow src-burrow ... Clear target, and adds units from source burrows.

add-units target-burrow src-burrow ... Add units from the source burrows to the target.

remove-units target-burrow src-burrow ... Remove units in source burrows from the target.

set-tiles target-burrow src-burrow ... Clear target and adds tiles from the source burrows.

add-tiles target-burrow src-burrow ... Add tiles from the source burrows to the target.

remove-tiles target-burrow src-burrow ... Remove tiles in source burrows from the target.

For these three options, in place of a source burrow it is possible to use one of the following keywords: ABOVE_GROUND, SUBTERRANEAN, INSIDE, OUTSIDE, LIGHT, DARK, HIDDEN, REVEALED

Features:

auto-grow When a wall inside a burrow with a name ending in '+' is dug out, the burrow is extended to newly-revealed adjacent walls. This final '+' may be omitted in burrow name args of commands above. Digging 1-wide corridors with the miner inside the burrow is SLOW.

changelayer

Changes material of the geology layer under cursor to the specified inorganic RAW material. Can have impact on all surrounding regions, not only your embark! By default changing stone to soil and vice versa is not allowed. By default changes only the layer at the cursor position. Note that one layer can stretch across lots of z levels. By default changes only the geology which is linked to the biome under the cursor. That geology might be linked to other biomes as well, though. Mineral veins and gem clusters will stay on the map. Use *changevein* for them.

tl;dr: You will end up with changing quite big areas in one go, especially if you use it in lower z levels. Use with care.

Options:

all_biomes Change selected layer for all biomes on your map. Result may be undesirable since the same layer can AND WILL be on different z-levels for different biomes. Use the tool 'probe' to get an idea how layers and biomes are distributed on your map.

all_layers Change all layers on your map (only for the selected biome unless 'all_biomes' is added). Candy mountain, anyone? Will make your map quite boring, but tidy.

force Allow changing stone to soil and vice versa. !!THIS CAN HAVE WEIRD EFFECTS, USE WITH CARE!! Note that soil will not be magically replaced with stone. You will, however, get a stone floor after digging so it will allow the floor to be engraved. Note that stone will not be magically replaced with soil. You will, however, get a soil floor after digging so it could be helpful for creating farm plots on maps with no soil.

verbose Give some details about what is being changed.

trouble Give some advice about known problems.

Examples:

changelayer GRANITE Convert layer at cursor position into granite.

changelayer SILTY_CLAY force Convert layer at cursor position into clay even if it's stone.

changelayer MARBLE all_biomes all_layers Convert all layers of all biomes which are not soil into marble.

Note:

- If you use changelayer and nothing happens, try to pause/unpause the game for a while and try to move the cursor to another tile. Then try again. If that doesn't help try temporarily changing some other layer, undo your changes and try again for the layer you want to change. Saving and reloading your map might also help.
- You should be fine if you only change single layers without the use of 'force'. Still it's advisable to save your game before messing with the map.
- When you force changelayer to convert soil to stone you might experience weird stuff (flashing tiles, tiles changed all over place etc). Try reverting the changes manually or even better use an older savegame. You did save your game, right?

---

changevein

Changes material of the vein under cursor to the specified inorganic RAW material. Only affects tiles within the current 16x16 block - for veins and large clusters, you will need to use this command multiple times.

Example:

changevein NATIVE_PLATINUM Convert vein at cursor position into platinum ore.

changeitem

Allows changing item material and base quality. By default the item currently selected in the UI will be changed (you can select items in the 'k' list or inside containers/inventory). By default change is only allowed if materials is of the same subtype (for example wood<->wood, stone<->stone etc). But since some transformations work pretty well and may be desired you can override this with 'force'. Note that some attributes will not be touched, possibly resulting in weirdness. To get an idea how the RAW id should look like, check some items with 'info'. Using 'force' might create items which are not touched by crafters/haulers.

Options:

info Don't change anything, print some info instead.

here Change all items at the cursor position. Requires in-game cursor.

material, m Change material. Must be followed by valid material RAW id.

quality, q Change base quality. Must be followed by number (0-5).

force Ignore subtypes, force change to new material.

Examples:

changeitem m INORGANIC:GRANITE here Change material of all items under the cursor to granite.

changeitem q 5 Change currently selected item to masterpiece quality.

cleanconst

Cleans up construction materials.

This utility alters all constructions on the map so that they spawn their building component when they are disassembled, allowing their actual build items to be safely deleted. This can improve FPS in extreme situations.

deramp

Removes all ramps designated for removal from the map. This is useful for replicating the old channel digging designation. It also removes any and all 'down ramps' that can remain after a cave-in (you don't have to designate anything for that to happen).

dig

This plugin makes many automated or complicated dig patterns easy.

Basic commands:

digv Designate all of the selected vein for digging.

digvx Also cross z-levels, digging stairs as needed. Alias for `digv x`.

digl Like `digv`, for layer stone. Also supports an `undo` option to remove designations, for if you accidentally set 50 levels at once.

diglx Also cross z-levels, digging stairs as needed. Alias for `digl x`.

digexp

This command is for exploratory mining.

There are two variables that can be set: pattern and filter.

Patterns:

diag5 diagonals separated by 5 tiles

diag5r diag5 rotated 90 degrees

ladder A 'ladder' pattern

ladderr ladder rotated 90 degrees

clear Just remove all dig designations

cross A cross, exactly in the middle of the map.

Filters:

all designate whole z-level

hidden designate only hidden tiles of z-level (default)

designated Take current designation and apply pattern to it.

After you have a pattern set, you can use `expdig` to apply it again.

Examples:

`expdig diag5 hidden` Designate the diagonal 5 patter over all hidden tiles

`expdig` Apply last used pattern and filter

`expdig ladder designated` Take current designations and replace them with the ladder pattern

digcircle

A command for easy designation of filled and hollow circles. It has several types of options.

Shape:

hollow Set the circle to hollow (default)

filled Set the circle to filled

Diameter in tiles (default = 0, does nothing)

Action:

set Set designation (default)

unset Unset current designation

invert Invert designations already present

Designation types:

dig Normal digging designation (default)

ramp Ramp digging

ustair Staircase up

dstair Staircase down

xstair Staircase up/down

chan Dig channel

After you have set the options, the command called with no options repeats with the last selected parameters.

Examples:

`digcircle filled 3` Dig a filled circle with diameter = 3.

`digcircle` Do it again.

digtype

For every tile on the map of the same vein type as the selected tile, this command designates it to have the same designation as the selected tile. If the selected tile has no designation, they will be dig designated. If an argument is given, the designation of the selected tile is ignored, and all appropriate tiles are set to the specified designation.

Options:

dig

channel

ramp

updown up/down stairs

up up stairs

down down stairs

clear clear designation

digFlood

Automatically digs out specified veins as they are discovered. It runs once every time a dwarf finishes a dig job. It will only dig out appropriate tiles that are adjacent to the finished dig job. To add a vein type, use `digFlood 1 [type]`. This will also enable the plugin. To remove a vein type, use `digFlood 0 [type] 1` to disable, then remove, then re-enable.

Usage:

help digflood detailed help message

digFlood 0 disable the plugin

digFlood 1 enable the plugin

digFlood 0 MICROCLINE COAL_BITUMINOUS 1 disable plugin, remove microcline and bituminous coal from monitoring, then re-enable the plugin

digFlood CLEAR remove all inorganics from monitoring

digFlood digAll1 ignore the monitor list and dig any vein

digFlood digAll0 disable digAll mode

filltraffic

Set traffic designations using flood-fill starting at the cursor. See also *alltraffic*, *restrictice*, and *restrictliquids*. Options:

H High Traffic

N Normal Traffic

L Low Traffic

R Restricted Traffic

X Fill across z-levels.

B Include buildings and stockpiles.

P Include empty space.

Example:

`filltraffic H` When used in a room with doors, it will set traffic to HIGH in just that room.

fortplan

Usage: `fortplan [filename]`

Designates furniture for building according to a `.csv` file with quickfort-style syntax. Companion to *digfort*.

The first line of the file must contain the following:

```
#build start(X; Y; <start location description>)
```

...where X and Y are the offset from the top-left corner of the file's area where the in-game cursor should be located, and `<start location description>` is an optional description of where that is. You may also leave a description of the contents of the file itself following the closing parenthesis on the same line.

The syntax of the file itself is similar to *digfort* or quickfort. At present, only buildings constructed of an item with the same name as the building are supported. All other characters are ignored. For example:

```
`,`,d,`,`
`,f,`,t,`
`,s,b,c,`
```

This section of a file would designate for construction a door and some furniture inside a bedroom: specifically, clockwise from top left, a cabinet, a table, a chair, a bed, and a statue.

All of the building designation uses *buildingplan*, so you do not need to have the items available to construct all the buildings when you run fortplan with the .csv file.

getplants

This tool allows plant gathering and tree cutting by RAW ID. Specify the types of trees to cut down and/or shrubs to gather by their plant names, separated by spaces.

Options:

-t Select trees only (exclude shrubs)

-s Select shrubs only (exclude trees)

-c Clear designations instead of setting them

-x Apply selected action to all plants except those specified (invert selection)

-a Select every type of plant (obeys `-t`/`-s`)

Specifying both `-t` and `-s` will have no effect. If no plant IDs are specified, all valid plant IDs will be listed.

infiniteSky

Automatically allocates new z-levels of sky at the top of the map as you build up, or on request allocates many levels all at once.

Usage:

`infiniteSky n` Raise the sky by n z-levels.

`infiniteSky enable/disable` Enables/disables monitoring of constructions. If you build anything in the second to highest z-level, it will allocate one more sky level. This is so you can continue to build stairs upward.

Sometimes new z-levels disappear and cause cave-ins. Saving and loading after creating new z-levels should fix the problem.

liquids

Allows adding magma, water and obsidian to the game. It replaces the normal dfhack command line and can't be used from a hotkey. Settings will be remembered as long as dfhack runs. Intended for use in combination with the command `liquids-here` (which can be bound to a hotkey). See also Issue 80.

Note: Spawning and deleting liquids can mess up pathing data and temperatures (creating heat traps). You've been warned.

Settings will be remembered until you quit DF. You can call *liquids-here* to execute the last configured action, which is useful in combination with keybindings.

Usage: point the DF cursor at a tile you want to modify and use the commands.

If you only want to add or remove water or magma from one tile, *source* may be easier to use.

Commands

Misc commands:

q quit

help, ? print this list of commands

<empty line> put liquid

Modes:

m switch to magma

w switch to water

o make obsidian wall instead

of make obsidian floors

rs make a river source

f flow bits only

wclean remove salt and stagnant flags from tiles

Set-Modes and flow properties (only for magma/water):

s+ only add mode

s. set mode

s- only remove mode

f+ make the spawned liquid flow

f. don't change flow state (read state in flow mode)

f- make the spawned liquid static

Permaflow (only for water):

pf. don't change permaflow state

pf- make the spawned liquid static

pf[NS][EW] make the spawned liquid permanently flow

0-7 set liquid amount

Brush size and shape:

p, point Single tile

r, range Block with cursor at bottom north-west (any place, any size)

block DF map block with cursor in it (regular spaced 16x16x1 blocks)

column Column from cursor, up through free space

flood Flood-fill water tiles from cursor (only makes sense with wclean)

liquids-here

Run the liquid spawner with the current/last settings made in liquids (if no settings in liquids were made it paints a point of 7/7 magma by default).

Intended to be used as keybinding. Requires an active in-game cursor.

plant

A tool for creating shrubs, growing, or getting rid of them.

Subcommands:

create Creates a new sapling under the cursor. Takes a raw ID as argument (e.g. TOWER_CAP). The cursor must be located on a dirt or grass floor tile.

grow Turns saplings into trees; under the cursor if a sapling is selected, or every sapling on the map if the cursor is hidden.

extirpate Kills the tree or shrub under the cursor, instantly turning them to ashes.

immolate Sets the plants on fire instead. The fires can and *will* spread ;)

For mass effects, use one of the additional options:

shrubs affect all shrubs on the map

trees affect all trees on the map

all affect every plant!

regrass

Regrows all the grass. Not much to it ;)

restrictice

Restrict traffic on all tiles on top of visible ice. See also *alltraffic*, *filltraffic*, and *restrictliquids*.

restrictliquids

Restrict traffic on all visible tiles with liquid. See also *alltraffic*, *filltraffic*, and *restrictice*.

tiletypes

Can be used for painting map tiles and is an interactive command, much like *liquids*. If something goes wrong, *fixveins* may help.

The tool works with two set of options and a brush. The brush determines which tiles will be processed. First set of options is the filter, which can exclude some of the tiles from the brush by looking at the tile properties. The second set of options is the paint - this determines how the selected tiles are changed.

Both paint and filter can have many different properties including things like general shape (WALL, FLOOR, etc.), general material (SOIL, STONE, MINERAL, etc.), state of 'designated', 'hidden' and 'light' flags.

The properties of filter and paint can be partially defined. This means that you can for example turn all stone fortifications into floors, preserving the material:

```
filter material STONE
filter shape FORTIFICATION
paint shape FLOOR
```

Or turn mineral vein floors back into walls:

```
filter shape FLOOR
filter material MINERAL
paint shape WALL
```

The tool also allows tweaking some tile flags:

```
paint hidden 1
paint hidden 0
```

This will hide previously revealed tiles (or show hidden with the 0 option).

More recently, the tool supports changing the base material of the tile to an arbitrary stone from the raws, by creating new veins as required. Note that this mode paints under ice and constructions, instead of overwriting them. To enable, use:

```
paint stone MICROCLINE
```

This mode is incompatible with the regular `material` setting, so changing it cancels the specific stone selection:

```
paint material ANY
```

Since different vein types have different drop rates, it is possible to choose which one to use in painting:

```
paint veintype CLUSTER_SMALL
```

When the chosen type is `CLUSTER` (the default), the tool may automatically choose to use layer stone or lava stone instead of veins if its material matches the desired one.

Any paint or filter option (or the entire paint or filter) can be disabled entirely by using the ANY keyword:

```
paint hidden ANY
paint shape ANY
filter material any
filter shape any
filter any
```

You can use several different brushes for painting tiles:

point a single tile

range a rectangular range

column a column ranging from current cursor to the first solid tile above

block a DF map block - 16x16 tiles, in a regular grid

Example:

```
range 10 10 1
```

This will change the brush to a rectangle spanning 10x10 tiles on one z-level. The range starts at the position of the cursor and goes to the east, south and up.

For more details, use `tiletypes help`.

tiletypes-command

Runs tiletypes commands, separated by ;. This makes it possible to change tiletypes modes from a hotkey or via dfhack-run.

tiletypes-here

Apply the current tiletypes options at the in-game cursor position, including the brush. Can be used from a hotkey.

tiletypes-here-point

Apply the current tiletypes options at the in-game cursor position to a single tile. Can be used from a hotkey.

tubefill

Fills all the adamantine veins again. Veins that were hollow will be left alone.

Options:

hollow fill in naturally hollow veins too

Beware that filling in hollow veins will trigger a demon invasion on top of your miner when you dig into the region that used to be hollow.

2.3.6 Mods and Cheating

add-spatter

This plugin makes reactions with names starting with `SPATTER_ADD_` produce contaminants on the items instead of improvements. The plugin is intended to give some use to all those poisons that can be bought from caravans, so they're immune to being washed away by water or destroyed by *clean*.

adv-bodyswap

This allows taking control over your followers and other creatures in adventure mode. For example, you can make them pick up new arms and armor and equip them properly.

Usage:

- When viewing unit details, body-swaps into that unit.
- In the main adventure mode screen, reverts transient swap.

catsplosion

Makes cats just *multiply*. It is not a good idea to run this more than once or twice.

createitem

Allows creating new items of arbitrary types and made of arbitrary materials. By default, items created are spawned at the feet of the selected unit.

Specify the item and material information as you would indicate them in custom reaction raws, with the following differences:

- Separate the item and material with a space rather than a colon
- If the item has no subtype, omit the :NONE
- If the item is REMAINS, FISH, FISH_RAW, VERMIN, PET, or EGG, specify a CREATURE:CASTE pair instead of a material token.

Corpses, body parts, and prepared meals cannot be created using this tool.

Examples:

```
createitem GLOVES:ITEM_GLOVES_GAUNTLETS INORGANIC:STEEL 2
        Create 2 pairs of steel gauntlets.
createitem WOOD PLANT_MAT:TOWER_CAP:WOOD
        Create tower-cap logs.
```

To change where new items are placed, first run the command with a destination type while an appropriate destination is selected.

Options:

floor Subsequent items will be placed on the floor beneath the selected unit's feet.

item Subsequent items will be stored inside the currently selected item.

building Subsequent items will become part of the currently selected building. Good for loading traps; do not use with workshops (or deconstruct to use the item).

diggingInvaders

Makes invaders dig or destroy constructions to get to your dwarves.

To enable/disable the pluging, use: `diggingInvaders (1|enable)|(0|disable)`

Basic usage:

add GOBLIN registers the race GOBLIN as a digging invader. Case-sensitive.

remove GOBLIN unregisters the race GOBLIN as a digging invader. Case-sensitive.

now makes invaders try to dig now, if plugin is enabled

clear clears all digging invader races

edgesPerTick n makes the pathfinding algorithm work on at most n edges per tick. Set to 0 or lower to make it unlimited.

You can also use `diggingInvaders setCost (race) (action) n` to set the pathing cost of particular action, or `setDelay` to set how long it takes. Costs and delays are per-tile, and the table shows default values.

Action	Cost	Delay	Notes
`walk`	1	0	base cost in the path algorithm
`destroyBuilding`	2	1,000	delay adds to the job_completion_timer of destroy building jobs that are assigned to invaders
`dig`	10,000	1,000	digging soil or natural stone
`destroyRoughConstruction`	1,000	1,000	constructions made from boulders
`destroySmoothConstruction`	100	100	constructions made from blocks or bars

fastdwarf

Controls speedydwarf and teledwarf. Speedydwarf makes dwarves move quickly and perform tasks quickly. Teledwarf makes dwarves move instantaneously, but do jobs at the same speed.

fastdwarf 0 disables both (also `0 0`)

fastdwarf 1 enables speedydwarf and disables teledwarf (also `1 0`)

fastdwarf 2 sets a native debug flag in the game memory that implements an even more aggressive version of speedydwarf.

fastdwarf 0 1 disables speedydwarf and enables teledwarf

fastdwarf 1 1 enables both

See *superdwarf* for a per-creature version.

forceequip

Forceequip moves local items into a unit's inventory. It is typically used to equip specific clothing/armor items onto a dwarf, but can also be used to put armor onto a war animal or to add unusual items (such as crowns) to any unit.

For more information run `forceequip help`. See also *modtools/equip-item*.

lair

This command allows you to mark the map as a monster lair, preventing item scatter on abandon. When invoked as `lair reset`, it does the opposite.

Unlike *reveal*, this command doesn't save the information about tiles - you won't be able to restore state of real monster lairs using `lair reset`.

Options:

lair Mark the map as monster lair

lair reset Mark the map as ordinary (not lair)

mode

This command lets you see and change the game mode directly.

> **Warning:** Only use `mode` after making a backup of your save!
> Not all combinations are good for every situation and most of them will produce undesirable results. There are a few good ones though.

Examples:

- You are in fort game mode, managing your fortress and paused.
- You switch to the arena game mode, *assume control of a creature* and then
- switch to adventure game mode(1). You just lost a fortress and gained an adventurer. Alternatively:
- You are in fort game mode, managing your fortress and paused at the esc menu.
- You switch to the adventure game mode, assume control of a creature, then save or retire.
- You just created a returnable mountain home and gained an adventurer.

strangemood

Creates a strange mood job the same way the game itself normally does it.

Options:

-force Ignore normal strange mood preconditions (no recent mood, minimum moodable population, artifact limit not reached).

-unit Make the strange mood strike the selected unit instead of picking one randomly. Unit eligibility is still enforced.

-type <T> Force the mood to be of a particular type instead of choosing randomly based on happiness. Valid values for Tare "fey", "secretive", "possessed", "fell", and "macabre".

-skill S Force the mood to use a specific skill instead of choosing the highest moodable skill. Valid values are "miner", "carpenter", "engraver", "mason", "tanner", "weaver", "clothier", "weaponsmith", "armorsmith", "metalsmith", "gemcutter", "gemsetter", "woodcrafter", "stonecrafter", "metalcrafter", "glassmaker", "leatherworker", "bonecarver", "bowyer", and "mechanic".

Known limitations: if the selected unit is currently performing a job, the mood will not be started.

siege-engine

Siege engines in DF haven't been updated since the game was 2D, and can only aim in four directions. To make them useful above-ground, this plugin allows you to:

- link siege engines to stockpiles
- restrict operator skill levels (like workshops)
- load any object into a catapult, not just stones
- aim at a rectangular area in any direction, and across Z-levels

The front-end is implemented by *gui/siege-engine*.

power-meter

The power-meter plugin implements a modified pressure plate that detects power being supplied to gear boxes built in the four adjacent N/S/W/E tiles.

The configuration front-end is implemented by *gui/power-meter*.

steam-engine

The steam-engine plugin detects custom workshops with STEAM_ENGINE in their token, and turns them into real steam engines.

The vanilla game contains only water wheels and windmills as sources of power, but windmills give relatively little power, and water wheels require flowing water, which must either be a real river and thus immovable and limited in supply, or actually flowing and thus laggy.

Compared to the water reactor exploit, steam engines make a lot of sense!

Construction

The workshop needs water as its input, which it takes via a passable floor tile below it, like usual magma workshops do. The magma version also needs magma.

Due to DFHack limits, the workshop will collapse over true open space. However down stairs are passable but support machines, so you can use them.

After constructing the building itself, machines can be connected to the edge tiles that look like gear boxes. Their exact position is extracted from the workshop raws.

Like with collapse above, due to DFHack limits the workshop can only immediately connect to machine components built AFTER it. This also means that engines cannot be chained without intermediate axles built after both engines.

Operation

In order to operate the engine, queue the Stoke Boiler job (optionally on repeat). A furnace operator will come, possibly bringing a bar of fuel, and perform it. As a result, a "boiling water" item will appear in the `t` view of the workshop.

Note: The completion of the job will actually consume one unit of the appropriate liquids from below the workshop. This means that you cannot just raise 7 units of magma with a piston and have infinite power. However, liquid consumption should be slow enough that water can be supplied by a pond zone bucket chain.

Every such item gives 100 power, up to a limit of 300 for coal, and 500 for a magma engine. The building can host twice that amount of items to provide longer autonomous running. When the boiler gets filled to capacity, all queued jobs are suspended; once it drops back to 3+1 or 5+1 items, they are re-enabled.

While the engine is providing power, steam is being consumed. The consumption speed includes a fixed 10% waste rate, and the remaining 90% are applied proportionally to the actual load in the machine. With the engine at nominal 300 power with 150 load in the system, it will consume steam for actual 300*(10% + 90%*150/300) = 165 power.

Masterpiece mechanism and chain will decrease the mechanical power drawn by the engine itself from 10 to 5. Masterpiece barrel decreases waste rate by 4%. Masterpiece piston and pipe decrease it by further 4%, and also decrease the whole steam use rate by 10%.

Explosions

The engine must be constructed using barrel, pipe and piston from fire-safe, or in the magma version magma-safe metals.

During operation weak parts get gradually worn out, and eventually the engine explodes. It should also explode if toppled during operation by a building destroyer, or a tantruming dwarf.

Save files

It should be safe to load and view engine-using fortresses from a DF version without DFHack installed, except that in such case the engines won't work. However actually making modifications to them, or machines they connect to (including by pulling levers), can easily result in inconsistent state once this plugin is available again. The effects may be as weird as negative power being generated.

2.4 DFHack Scripts

Lua or ruby scripts placed in the `hack/scripts/` directory are considered for execution as if they were native DFHack commands.

The following pages document all the scripts in the DFHack standard library.

2.4.1 Basic Scripts

Basic scripts are not stored in any subdirectory, and can be invoked directly. They are generally useful tools for any player.

Contents

adaptation

View or set level of cavern adaptation for the selected unit or the whole fort. Usage: `adaptation <show|set> <him|all> [value]`. The `value` must be between 0 and 800,000 inclusive.

add-thought

Adds a thought or emotion to the selected unit. Can be used by other scripts, or the gui invoked by running `add-thought gui` with a unit selected.

adv-max-skills

When creating an adventurer, raises all changeable skills and attributes to their maximum level.

armoks-blessing

Runs the equivalent of *rejuvenate*, *elevate-physical*, *elevate-mental*, and *brainwash* on all dwarves currently on the map. This is an extreme change, which sets every stat to an ideal - legendary skills, great traits, and easy-to-satisfy preferences.

Without arguments, all attributes, age & personalities are adjusted. Arguments allow for skills to be adjusted as well.

autofarm

Automatically handle crop selection in farm plots based on current plant stocks. Selects a crop for planting if current stock is below a threshold. Selected crops are dispatched on all farmplots.

Usage:

```
autofarm start
autofarm default 30
autofarm threshold 150 helmet_plump tail_pig
```

autolabor-artisans

Runs an *autolabor* command, for all labors where skill level influences output quality. Examples:

```
autolabor-artisans 0 2 3
autolabor-artisans disable
```

autounsuspend

Automatically unsuspend jobs in workshops, on a recurring basis. See *unsuspend* for one-off use, or *resume* `all`.

ban-cooking

A more convenient way to ban cooking various categories of foods than the kitchen interface. Usage: `ban-cooking <type>`. Valid types are `booze`, `honey`, `tallow`, `oil`, `seeds` (non-tree plants with seeds), `brew`, `mill`, `thread`, and `milk`.

binpatch

Implements functions for in-memory binpatches. See *Patching the DF binary*.

brainwash

Modify the personality traits of the selected dwarf to match an 'ideal' personality - as stable and reliable as possible. This makes dwarves very stable, preventing tantrums even after months of misery.

burial

Sets all unowned coffins to allow burial. `burial -pets` also allows burial of pets.

colonies

List vermin colonies, place honey bees, or convert all vermin to honey bees. Usage:

colonies List all vermin colonies on the map.

colonies place Place a honey bee colony under the cursor.

colonies convert Convert all existing colonies to honey bees.

The `place` and `convert` subcommands by default create or convert to honey bees, as this is the most commonly useful. However both accept an optional flag to use a different vermin type, for example `colonies place ANT` creates an ant colony and `colonies convert TERMITE` ends your beekeeping industry.

combine-drinks

Merge stacks of drinks in the selected stockpile.

combine-plants

Merge stacks of plants or plant growths in the selected container or stockpile.

create-items

Spawn items under the cursor, to get your fortress started.

The first argument gives the item category, the second gives the material, and the optionnal third gives the number of items to create (defaults to 20).

Currently supported item categories: `boulder`, `bar`, `plant`, `log`, `web`.

Instead of material, using `list` makes the script list eligible materials.

The `web` item category will create an uncollected cobweb on the floor.

Note that the script does not enforce anything, and will let you create boulders of toad blood and stuff like that. However the `list` mode will only show 'normal' materials.

Examples:

```
create-items boulders COAL_BITUMINOUS 12
create-items plant tail_pig
create-items log list
create-items web CREATURE:SPIDER_CAVE_GIANT:SILK
create-items bar CREATURE:CAT:SOAP
create-items bar adamantine
```

deathcause

Select a body part ingame, or a unit from the `u` unit list, and this script will display the cause of death of the creature.

deterioratecloth es

Somewhere between a "mod" and a "fps booster", with a small impact on vanilla gameplay. All of those slightly worn wool shoes that dwarves scatter all over the place will deteriorate at a greatly increased rate, and eventually just crumble into nothing. As warm and fuzzy as a dining room full of used socks makes your dwarves feel, your FPS does not like it.

Usage: `deterioratecloth es (start|stop)`

deterioratecorpses

Somewhere between a "mod" and a "fps booster", with a small impact on vanilla gameplay.

In long running forts, especially evil biomes, you end up with a lot of toes, teeth, fingers, and limbs scattered all over the place. Various corpses from various sieges, stray kitten corpses, probably some heads. Basically, your map will look like a giant pile of assorted body parts, all of which individually eat up a small part of your FPS, which collectively eat up quite a bit.

In addition, this script also targets various butchery byproducts. Enjoying your thriving animal industry? Your FPS does not. Those thousands of skulls, bones, hooves, and wool eat up precious FPS that could be used to kill goblins and elves. Whose corpses will also get destroyed by the script to kill more goblins and elves.

This script causes all of those to rot away into nothing after several months.

Usage: `deterioratecorpses (start|stop)`

deteriorate food

Somewhere between a "mod" and a "fps booster", with a small impact on vanilla gameplay.

With this script running, all food and plants wear out and disappear after several months. Barrels and stockpiles will keep them from rotting, but it won't keep them from decaying. No more sitting on a hundred years worth of food. No more keeping barrels of pig tails sitting around until you decide to use them. Either use it, eat it, or lose it. Seeds, are excluded from this, if you aren't planning on using your pig tails, hold onto the seeds for a rainy day.

This script is...pretty far reaching. However, almost all long running forts I've had end up sitting on thousands and thousands of food items. Several thousand cooked meals, three thousand plump helmets, just as many fish and meat. It gets pretty absurd. And your FPS doesn't like it.

Usage: `deterioratefood (start|stop)`

digfort

A script to designate an area for digging according to a plan in csv format.

This script, inspired from quickfort, can designate an area for digging. Your plan should be stored in a .csv file like this:

```
# this is a comment
d;d;u;d;d;skip this tile;d
d;d;d;i
```

Available tile shapes are named after the 'dig' menu shortcuts: `d` for dig, `u` for upstairs, `j` downstairs, `i` updown, `h` channel, `r` upward ramp, `x` remove designation. Unrecognized characters are ignored (eg the 'skip this tile' in the sample).

Empty lines and data after a `#` are ignored as comments. To skip a row in your design, use a single `;`.

One comment in the file may contain the phrase `start(3,5)`. It is interpreted as an offset for the pattern: instead of starting at the cursor, it will start 3 tiles left and 5 tiles up from the cursor.

The script takes the plan filename, starting from the root df folder (where `Dwarf Fortress.exe` is found).

drain-aquifer

Remove all 'aquifer' tag from the map blocks. Irreversible.

elevate-mental

Set all mental attributes of the selected dwarf to 2600, which is very high. Numbers between 0 and 5000 can be passed as an argument: `elevate-mental 100` for example would make the dwarf very stupid indeed.

elevate-physical

As for elevate-mental, but for physical traits. High is good for soldiers, while having an ineffective hammerer can be useful too...

embark-skills

Adjusts dwarves' skills when embarking.

Note that already-used skill points are not taken into account or reset.

points N Sets the skill points remaining of the selected dwarf to `N`.

points N all Sets the skill points remaining of all dwarves to `N`.

max Sets all skills of the selected dwarf to "Proficient".

max all Sets all skills of all dwarves to "Proficient".

legendary Sets all skills of the selected dwarf to "Legendary".

legendary all Sets all skills of all dwarves to "Legendary".

emigration

Allows dwarves to emigrate from the fortress when stressed, in proportion to how badly stressed they are and adjusted for who they would have to leave with - a dwarven merchant being more attractive than leaving alone (or with an elf). The check is made monthly.

A happy dwarf (ie with negative stress) will never emigrate.

Usage: `emigration enable|disable`

exportlegends

Controls legends mode to export data - especially useful to set-and-forget large worlds, or when you want a map of every site when there are several hundred.

The 'info' option exports more data than is possible in vanilla, to a `region-date-legends_plus.xml` file developed to extend World Viewer and other legends utilities.

Options:

info Exports the world/gen info, the legends XML, and a custom XML with more information

custom Exports a custom XML with more information

sites Exports all available site maps

maps Exports all seventeen detailed maps

all Equivalent to calling all of the above, in that order

exterminate

Kills any unit of a given race.

With no argument, lists the available races and count eligible targets.

With the special argument `him`, targets only the selected creature.

With the special argument `undead`, targets all undeads on the map, regardless of their race.

When specifying a race, a caste can be specified to further restrict the targeting. To do that, append and colon and the caste name after the race.

Any non-dead non-caged unit of the specified race gets its `blood_count` set to 0, which means immediate death at the next game tick. For creatures such as vampires, it also sets animal.vanish_countdown to 2.

An alternate mode is selected by adding a 2nd argument to the command, `magma`. In this case, a column of 7/7 magma is generated on top of the targets until they die (Warning: do not call on magma-safe creatures. Also, using this mode on birds is not recommended.) The final alternate mode is `butcher`, which marks them for butchering but does not kill.

Will target any unit on a revealed tile of the map, including ambushers, but ignore caged/chained creatures.

Ex:

```
exterminate gob
exterminate gob:male
```

To kill a single creature, select the unit with the 'v' cursor and:

```
exterminate him
```

To purify all elves on the map with fire (may have side-effects):

```
exterminate elve magma
```

feature

Enables management of map features.

- Discovering a magma feature (magma pool, volcano, magma sea, or curious underground structure) permits magma workshops and furnaces to be built.
- Discovering a cavern layer causes plants (trees, shrubs, and grass) from that cavern to grow within your fortress.

Options:

list Lists all map features in your current embark by index.

magma Enable magma furnaces (discovers a random magma feature).

show X Marks the selected map feature as discovered.

hide X Marks the selected map feature as undiscovered.

fix-ster

Utilizes the orientation tag to either fix infertile creatures or inflict infertility on creatures that you do not want to breed. Usage:

```
fix-ster [fert|ster] [all|animals|only:<creature>]
```

`fert` or `ster` is a required argument; whether to make the target fertile or sterile. Optional arguments specify the target: no argument for the selected unit, `all` for all units on the map, `animals` for all non-dwarf creatures, or `only:<creature>` to only process matching creatures.

fixnaked

Removes all unhappy thoughts due to lack of clothing.

forum-dwarves

Saves a copy of a text screen, formatted in bbcode for posting to the Bay12 Forums. Use `forum-dwarves help` for more information.

full-heal

Attempts to fully heal the selected unit. `full-heal -r` attempts to resurrect the unit.

gaydar

Shows the sexual orientation of units, useful for social engineering or checking the viability of livestock breeding programs. Use `gaydar -help` for information on available filters for orientation, citizenship, species, etc.

growcrops

Instantly grow seeds inside farming plots.

With no argument, this command list the various seed types currently in use in your farming plots. With a seed type, the script will grow 100 of these seeds, ready to be harvested. Set the number with a 2nd argument.

For example, to grow 40 plump helmet spawn:

```
growcrops plump 40
```

hfs-pit

Creates a pit to the underworld at the cursor.

Takes three arguments: diameter of the pit in tiles, whether to wall off the pit, and whether to insert stairs. If no arguments are given, the default is `hfs-pit 1 0 0`, ie single-tile wide with no walls or stairs.:

```
hfs-pit 4 0 1
hfs-pit 2 1 0
```

First example is a four-across pit with stairs but no walls; second is a two-across pit with stairs but no walls.

hotkey-notes

Lists the key, name, and jump position of your hotkeys in the DFHack console.

item-descriptions

Exports a table with custom description text for every item in the game. Used by *view-item-info*; see instructions there for how to override for mods.

launch

Activate with a cursor on screen and you will go there rapidly, attack something first to send them there.

lever

Allow manipulation of in-game levers from the dfhack console.

Can list levers, including state and links, with:

```
lever list
```

To queue a job so that a dwarf will pull the lever 42, use `lever pull 42`. This is the same as `q` querying the building and queue a `P` pull request.

To magically toggle the lever immediately, use:

```
lever pull 42 --now
```

locate-ore

Scan the map for metal ores.

Finds and designate for digging one tile of a specific metal ore. Only works for native metal ores, does not handle reaction stuff (eg STEEL).

When invoked with the `list` argument, lists metal ores available on the map.

Examples:

```
locate-ore list
locate-ore hematite
locate-ore iron
```

lua

There are the following ways to invoke this command:

1. `lua` (without any parameters)

   This starts an interactive lua interpreter.

2. `lua -f "filename"` or `lua --file "filename"`

   This loads and runs the file indicated by filename.

3. `lua -s ["filename"]` or `lua --save ["filename"]`

   This loads and runs the file indicated by filename from the save directory. If the filename is not supplied, it loads "dfhack.lua".

4. `:lua` *lua statement...*

   Parses and executes the lua statement like the interactive interpreter would.

make-legendary

Makes the selected dwarf legendary in one skill, a group of skills, or all skills. View groups with `make-legendary classes`, or all skills with `make-legendary list`. Use `make-legendary MINING` when you need something dug up, or `make-legendary all` when only perfection will do.

make-monarch

Make the selected unit King or Queen of your civilisation.

markdown

Save a copy of a text screen in markdown (for reddit among others). Use `markdown help` for more details.

masspit

Designate all creatures in cages on top of a pit/pond activity zone for pitting. Works best with an animal stockpile on top of the zone.

Works with a zone number as argument (eg `Activity Zone #6` -> `masspit 6`) or with the game cursor on top of the area.

migrants-now

Forces an immediate migrant wave. Only works after migrants have arrived naturally. Equivalent to *modtools/force* `-eventType migrants`

multicmd

Run multiple dfhack commands. The argument is split around the character ; and all parts are run sequentially as independent dfhack commands. Useful for hotkeys.

Example:

```
multicmd locate-ore IRON ; digv ; digcircle 16
```

names

Rename units or items. Usage:

-help print this help message

-item if viewing an item

-unit if viewing a unit

-first [Somename | "Some Names like This" if a first name is desired, leave blank to clear current first name

open-legends

Open a legends screen when in fortress mode. Compatible with *exportlegends*.

points

Sets available points at the embark screen to the specified number. Eg. `points 1000000` would allow you to buy everything, or `points 0` would make life quite difficult.

position

Reports the current time: date, clock time, month, and season. Also reports location: z-level, cursor position, window size, and mouse location.

pref-adjust

A two-stage script: `pref-adjust clear` removes preferences from all dwarves, and `pref-adjust` inserts an 'ideal' set which is easy to satisfy:

```
Feb Idashzefon likes wild strawberries for their vivid red color,
fisher berries for their round shape, prickle berries for their
precise thorns, plump helmets for their rounded tops, prepared meals,
plants, drinks, doors, thrones, tables and beds. When possible, she
prefers to consume wild strawberries, fisher berries, prickle
berries, plump helmets, strawberry wine, fisher berry wine, prickle
berry wine, and dwarven wine.
```

prefchange

Sets preferences for mooding to include a weapon type, equipment type, and material. If you also wish to trigger a mood, see *strangemood*.

Valid options:

show show preferences of all units

c clear preferences of selected unit

all clear preferences of all units

axp likes axes, breastplates, and steel

has likes hammers, mail shirts, and steel

swb likes short swords, high boots, and steel

spb likes spears, high boots, and steel

mas likes maces, shields, and steel

xbh likes crossbows, helms, and steel

pig likes picks, gauntlets, and steel

log likes long swords, gauntlets, and steel

dap likes daggers, greaves, and steel

Feel free to adjust the values as you see fit, change the has steel to platinum, change the axp axes to great axes, whatnot.

putontable

Makes item appear on the table, like in adventure mode shops. Arguments: `-a` or `--all` for all items.

quicksave

If called in dwarf mode, makes DF immediately saves the game by setting a flag normally used in seasonal auto-save.

region-pops

Show or modify the populations of animals in the region.

Usage:

region-pops list [pattern] Lists encountered populations of the region, possibly restricted by pattern.

region-pops list-all [pattern] Lists all populations of the region.

region-pops boost <TOKEN> <factor> Multiply all populations of TOKEN by factor. If the factor is greater than one, increases the population, otherwise decreases it.

region-pops boost-all <pattern> <factor> Same as above, but match using a pattern acceptable to list.

region-pops incr <TOKEN> <factor> Augment (or diminish) all populations of TOKEN by factor (additive).

region-pops incr-all <pattern> <factor> Same as above, but match using a pattern acceptable to list.

rejuvenate

Set the age of the selected dwarf to 20 years. Useful if valuable citizens are getting old, or there are too many babies around...

remove-stress

Sets stress to -1,000,000; the normal range is 0 to 500,000 with very stable or very stressed dwarves taking on negative or greater values respectively. Applies to the selected unit, or use `remove-stress -all` to apply to all units.

remove-wear

Sets the wear on all items in your fort to zero.

repeat

Repeatedly calls a lua script at the specified interval.

This allows neat background changes to the function of the game, especially when invoked from an init file. For detailed usage instructions, use `repeat -help`.

Usage examples:

```
repeat -name jim -time delay -timeUnits units -printResult true -command [ printArgs 3 1 2 ]
repeat -time 1 -timeUnits months -command [ multicmd cleanowned scattered x; clean all ] -name clean
```

The first example is abstract; the second will regularly remove all contaminants and worn items from the game.

`-name` sets the name for the purposes of cancelling and making sure you don't schedule the same repeating event twice. If not specified, it's set to the first argument after `-command`. `-time delay -timeUnits units`; delay is some positive integer, and units is some valid time unit for `dfhack.timeout(delay,timeUnits,function)`. `-command [ ... ]` specifies the command to be run.

setfps

Run `setfps <number>` to set the FPS cap at runtime, in case you want to watch combat in slow motion or something.

show-unit-syndromes

Show syndromes affecting units and the remaining and maximum duration, along with (optionally) substantial detail on the effects.

Use one or more of the following options:

help Show the help message

showall Show units even if not affected by any syndrome

showeffects Show detailed effects of each syndrome

showdisplayeffects Show effects that only change the look of the unit

selected Show selected unit

dwarves Show dwarves

livestock Show livestock

wildanimals Show wild animals

hostile Show hostiles (e.g. invaders, thieves, forgotten beasts etc)

world Show all defined syndromes in the world

export `export:<filename>` sends output to the given file, showing all syndromes affecting each unit with the maximum and present duration.

siren

Wakes up sleeping units, cancels breaks and stops parties either everywhere, or in the burrows given as arguments. In return, adds bad thoughts about noise, tiredness and lack of protection. Also, the units with interrupted breaks will go on break again a lot sooner. The script is intended for emergencies, e.g. when a siege appears, and all your military is partying.

source

Create an infinite magma or water source or drain on a tile. For more complex commands, try the *liquids* plugin.

This script registers a map tile as a liquid source, and every 12 game ticks that tile receives or remove 1 new unit of flow based on the configuration.

Place the game cursor where you want to create the source (must be a flow-passable tile, and not too high in the sky) and call:

```
source add [magma|water] [0-7]
```

The number argument is the target liquid level (0 = drain, 7 = source).

To add more than 1 unit everytime, call the command again on the same spot.

To delete one source, place the cursor over its tile and use `source delete`. To remove all existing sources, call `source clear`.

The `list` argument shows all existing sources.

Examples:

```
source add water     - water source
source add magma 7   - magma source
source add water 0   - water drain
```

spawnunit

`spawnunit RACE CASTE` creates a unit of the given race and caste at the cursor, by wrapping *modtools/create-unit*. Run `spawnunit help` for more options.

startdwarf

Use at the embark screen to embark with the specified number of dwarves. Eg. `startdwarf 500` would lead to a severe food shortage and FPS issues, while `startdwarf 10` would just allow a few more warm bodies to dig in. The number must be 7 or greater.

starvingdead

Somewhere between a "mod" and a "fps booster", with a small impact on vanilla gameplay. It mostly helps prevent undead cascades in the caverns, where constant combat leads to hundreds of undead roaming the caverns and destroying your FPS.

With this script running, all undead that have been on the map for one month gradually decay, losing strength, speed, and toughness. After six months, they collapse upon themselves, never to be reanimated.

Usage: `starvingdead (start|stop)`

stripcaged

For dumping items inside cages. Will mark selected items for dumping, then a dwarf may come and actually dump them (or you can use *autodump*).

Arguments:

list display the list of all cages and their item content on the console

items dump items in the cage, excluding stuff worn by caged creatures

weapons dump equipped weapons

armor dump everything worn by caged creatures (including armor and clothing)

all dump everything in the cage, on a creature or not

Without further arguments, all commands work on all cages and animal traps on the map. With the `here` argument, considers only the in-game selected cage (or the cage under the game cursor). To target only specific cages, you can alternatively pass cage IDs as arguments:

```
stripcaged weapons 25321 34228
```

superdwarf

Similar to *fastdwarf*, per-creature.

To make any creature superfast, target it ingame using 'v' and:

```
superdwarf add
```

Other options available: `del`, `clear`, `list`.

This script also shortens the 'sleeping' and 'on break' periods of targets.

teleport

Teleports a unit to given coordinates. Examples:

teleport -showunitid prints unitid beneath cursor

teleport -showpos prints coordinates beneath cursor

teleport -unit 1234 -x 56 -y 115 -z 26 teleports unit 1234 to 56,115,26

tidlers

Toggle between all possible positions where the idlers count can be placed.

troubleshoot-item

Print various properties of the selected item.

twaterlvl

Toggle between displaying/not displaying liquid depth as numbers.

undump-buildings

Undesignates building base materials for dumping.

unsuspend

Unsuspend jobs in workshops, on a one-off basis. See *autounsuspend* for regular use.

view-item-info

A script to extend the item or unit viewscreen with additional information including a custom description of each item (when available), and properties such as material statistics, weapon attacks, armor effectiveness, and more.

The associated script *item-descriptions*.lua supplies custom descriptions of items. Individual descriptions can be added or overridden by a similar script `raw/scripts/more-item-descriptions.lua`. Both work as sparse lists, so missing items simply go undescribed if not defined in the fallback.

warn-starving

If any (live) units are starving, very thirsty, or very drowsy, the game will be paused and a warning shown and logged to the console. Use with the *repeat* command for regular checks.

Use `warn-starving all` to display a list of all problematic units.

weather

Prints a map of the local weather, or with arguments `clear`, `rain`, and `snow` changes the weather.

2.4.2 Development Scripts

`devel/*` scripts are intended for developer use, but many may be of interest to anyone investigating odd phenomema or just messing around. They are documented to encourage such inquiry.

Some can PERMANENTLY DAMAGE YOUR SAVE if misused, so please be careful. The warnings are real; if in doubt make backups before running the command.

Contents

devel/all-bob

Changes the first name of all units to "Bob". Useful for testing *modtools/interaction-trigger* events.

devel/annc-monitor

Displays announcements and reports in the console.

enable|start Begins monitoring

disable|stop Stops monitoring

interval X Sets the delay between checks for new announcements to `X` frames

devel/check-release

Basic checks for release readiness

devel/clear-script-env

Clears the environment of the specified lua script(s).

devel/click-monitor

Displays the grid coordinates of mouse clicks in the console. Useful for plugin/script development.

Usage: `devel/click-monitor start|stop`

devel/cmptiles

Lists and/or compares two tiletype material groups.

Usage: `devel/cmptiles material1 [material2]`

devel/export-dt-ini

Exports an ini file containing memory addresses for Dwarf Therapist.

devel/find-offsets

WARNING: THIS SCRIPT IS STRICTLY FOR DFHACK DEVELOPERS.

Running this script on a new DF version will NOT MAKE IT RUN CORRECTLY if any data structures changed, thus possibly leading to CRASHES AND/OR PERMANENT SAVE CORRUPTION.

Finding the first few globals requires this script to be started immediately after loading the game, WITHOUT first loading a world. The rest expect a loaded save, not a fresh embark. Finding current_weather requires a special save previously processed with *devel/prepare-save* on a DF version with working dfhack.

The script expects vanilla game configuration, without any custom tilesets or init file changes. Never unpause the game unless instructed. When done, quit the game without saving using 'die'.

Arguments:

- global names to force finding them
- `all` to force all globals
- `nofeed` to block automated fake input searches
- `nozoom` to disable neighboring object heuristics

devel/inject-raws

WARNING: THIS SCRIPT CAN PERMANENLY DAMAGE YOUR SAVE.

This script attempts to inject new raw objects into your world. If the injected references do not match the actual edited raws, your save will refuse to load, or load but crash.

This script can handle reaction, item and building definitions.

The savegame contains a list of the relevant definition tokens in the right order, but all details are read from raws every time. This allows just adding stub definitions, and simply saving and reloading the game.

This is useful enough for modders and some users to justify the danger.

Usage example:

```
devel/inject-raws trapcomp ITEM_TRAPCOMP_STEAM_PISTON workshop STEAM_ENGINE MAGMA_STEAM_ENGINE react
```

devel/inspect-screen

Read the tiles from the screen and display info about them.

devel/light

An experimental lighting engine for DF, using the *rendermax* plugin.

Call `devel/light static` to not recalculate lighting when in game. Press ~ to recalculate lighting. Press ` to exit.

devel/list-filters

List input items for the building currently being built. This is where the filters in lua/dfhack/buildings.lua come from.

devel/lsmem

Prints memory ranges of the process.

devel/lua-example

An example lua script, which reports the number of times it has been called. Useful for testing environment persistence.

devel/modstate-monitor

Display changes in key modifier state, ie Ctrl/Alt/Shift.

enable|start Begin monitoring

disable|stop End monitoring

devel/nuke-items

Deletes ALL items not held by units, buildings or jobs. Intended solely for lag investigation.

devel/pop-screen

For killing bugged out gui script screens.

devel/prepare-save

WARNING: THIS SCRIPT IS STRICTLY FOR DFHACK DEVELOPERS.

This script prepares the current savegame to be used with *devel/find-offsets*. It CHANGES THE GAME STATE to predefined values, and initiates an immediate *quicksave*, thus PERMANENTLY MODIFYING the save.

devel/print-args

Prints all the arguments you supply to the script on their own line. Useful for debugging other scripts.

devel/print-args2

Prints all the arguments you supply to the script on their own line with quotes around them.

devel/save-version

Display DF version information about the current save

devel/scanitemother

List indices in `world.item.other[]` where current selected item appears.

devel/spawn-unit-helper

Setup stuff to allow arena creature spawn after a mode change.

With Arena spawn data initialized:

- enter the `k` menu and change mode using `rb_eval df.gametype = :DWARF_ARENA`
- spawn creatures (`c` ingame)
- revert to game mode using `rb_eval df.gametype = #{df.gametype.inspect}`
- To convert spawned creatures to livestock, select each one with the `v` menu, and enter `rb_eval df.unit_find.civ_id = df.ui.civ_id`

devel/test-perlin

Generates an image using multiple octaves of perlin noise.

devel/unforbidall

Unforbid all items.

devel/unit-path

Show the internal path a unit is currently following.

devel/watch-minecarts

Logs minecart coordinates and speeds to console.

Usage: `devel/watch-minecarts start|stop`

2.4.3 Bugfixing Scripts

`fix/*` scripts fix various bugs and issues, some of them obscure.

Contents

fix/blood-del

Makes it so that future caravans won't bring barrels full of blood, ichor, or goo.

fix/build-location

Fixes construction jobs that are stuck trying to build a wall while standing on the same exact tile (Bug 5991), designates the tile restricted traffic to hopefully avoid jamming it again, and unsuspends them.

fix/dead-units

Removes uninteresting dead units from the unit list. Doesn't seem to give any noticeable performance gain, but migrants normally stop if the unit list grows to around 3000 units, and this script reduces it back.

fix/dry-buckets

Removes water from all buckets in your fortress, allowing them to be used for making lye. Skips buckets in buildings (eg a well), being carried, or currently used by a job.

fix/fat-dwarves

Avoids 5-10% FPS loss due to constant recalculation of insulation for dwarves at maximum fatness, by reducing the cap from 1,000,000 to 999,999. Recalculation is triggered in steps of 250 units, and very fat dwarves constantly bounce off the maximum value while eating.

fix/feeding-timers

Reset the GiveWater and GiveFood timers of all units as appropriate.

fix/item-occupancy

Diagnoses and fixes issues with nonexistant 'items occupying site', usually caused by *autodump* bugs or other hacking mishaps. Checks that:

1. Item has `flags.on_ground` <=> it is in the correct block item list
2. A tile has items in block item list <=> it has `occupancy.item`
3. The block item lists are sorted

fix/loyaltycascade

Aborts loyalty cascades by fixing units whose own civ is the enemy.

fix/population-cap

Run this after every migrant wave to ensure your population cap is not exceeded.

The reason for population cap problems is that the population value it is compared against comes from the last dwarven caravan that successfully left for mountainhomes. This script instantly updates it. Note that a migration wave can still overshoot the limit by 1-2 dwarves because of the last migrant bringing his family. Likewise, king arrival ignores cap.

fix/stable-temp

Instantly sets the temperature of all free-lying items to be in equilibrium with the environment, which stops temperature updates until something changes. To maintain this efficient state, use *tweak fast-heat*.

fix/stuckdoors

Fix doors that are stuck open due to incorrect map occupancy flags, eg due to incorrect use of *teleport*.

2.4.4 GUI Scripts

`gui/*` scripts implement dialogs in the main game window.

In order to avoid user confusion, as a matter of policy all these tools display the word "DFHack" on the screen somewhere while active. When that is not appropriate because they merely add keybinding hints to existing DF screens, they deliberately use red instead of green for the key.

Contents

gui/advfort

This script allows to perform jobs in adventure mode. For more complete help press `?` while script is running. It's most comfortable to use this as a keybinding. (e.g. `keybinding set Ctrl-T gui/advfort`). Possible arguments:

-a, –nodfassign uses different method to assign items.

-i, –inventory checks inventory for possible items to use in the job.

-c, –cheat relaxes item requirements for buildings (e.g. walls from bones). Implies -a

job selects that job (e.g. Dig or FellTree)

An example of player digging in adventure mode:

WARNING: changes only persist in non procedural sites, namely: player forts, caves, and camps.

gui/advfort_items

Does something with items in adventure mode jobs.

gui/assign-rack

This script requires a binpatch, which has not been available since DF 0.34.11

See Bug 1445 for more info about the patches.

gui/autobutcher

An in-game interface for *autobutcher*.

gui/choose-weapons

Bind to a key (the example config uses `CtrlW`), and activate in the Equip->View/Customize page of the military screen.

Depending on the cursor location, it rewrites all 'individual choice weapon' entries in the selected squad or position to use a specific weapon type matching the assigned unit's top skill. If the cursor is in the rightmost list over a weapon entry, it rewrites only that entry, and does it even if it is not 'individual choice'.

Rationale: individual choice seems to be unreliable when there is a weapon shortage, and may lead to inappropriate weapons being selected.

gui/clone-uniform

Bind to a key (the example config uses `CtrlC`), and activate in the Uniforms page of the military screen with the cursor in the leftmost list.

When invoked, the script duplicates the currently selected uniform template, and selects the newly created copy.

gui/companion-order

A script to issue orders for companions. Select companions with lower case chars, issue orders with upper case. Must be in look or talk mode to issue command on tile.

- move - orders selected companions to move to location. If companions are following they will move no more than 3 tiles from you.
- equip - try to equip items on the ground.
- pick-up - try to take items into hand (also wield)
- unequip - remove and drop equipment
- unwield - drop held items
- wait - temporarily remove from party
- follow - rejoin the party after "wait"
- leave - remove from party (can be rejoined by talking)

gui/confirm-opts

A basic configuration interface for the *confirm* plugin.

gui/create-item

A graphical interface for creating items.

gui/dfstatus

Show a quick overview of critical stock quantities, including food, drinks, wood, and various bars. Sections can be enabled/disabled/configured by editing `dfhack-config/dfstatus.lua`.

gui/extended-status

Adds more subpages to the `z` status screen.

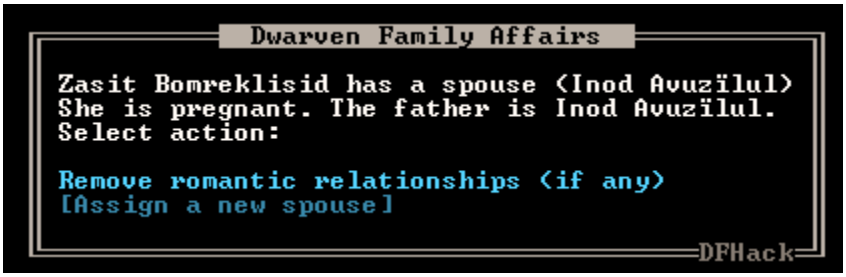
Usage:

```
gui/extended-status enable|disable|help|subpage_names
enable|disable gui/extended-status
```

gui/family-affairs

A user-friendly interface to view romantic relationships, with the ability to add, remove, or otherwise change them at your whim - fantastic for depressed dwarves with a dead spouse (or matchmaking players...).

The target/s must be alive, sane, and in fortress mode.



gui/family-affairs [unitID] shows GUI for the selected unit, or the specified unit ID

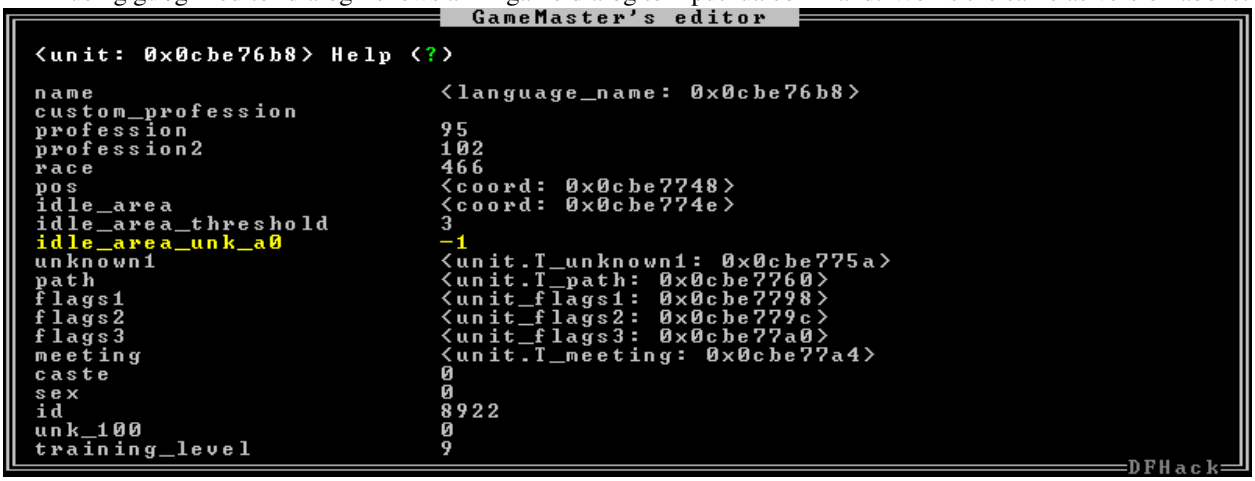
gui/family-affairs divorce [unitID] removes all spouse and lover information from the unit and it's partner, bypassing almost all checks.

gui/family-affairs [unitID] [unitID] divorces the two specificed units and their partners, then arranges for the two units to marry, bypassing almost all checks. Use with caution.

gui/gm-editor

This editor allows to change and modify almost anything in df. Press `?` for in-game help. There are three ways to open this editor:

- Callling `gui/gm-editor` from a command or keybinding opens the editor on whatever is selected or viewed (e.g. unit/item description screen)
- using gui/gm-editor <lua command> - executes lua command and opens editor on its results (e.g. `gui/gm-editor "df.global.world.items.all"` shows all items)
- using gui/gm-editor dialog - shows an in game dialog to input lua command. Works the same as version above.

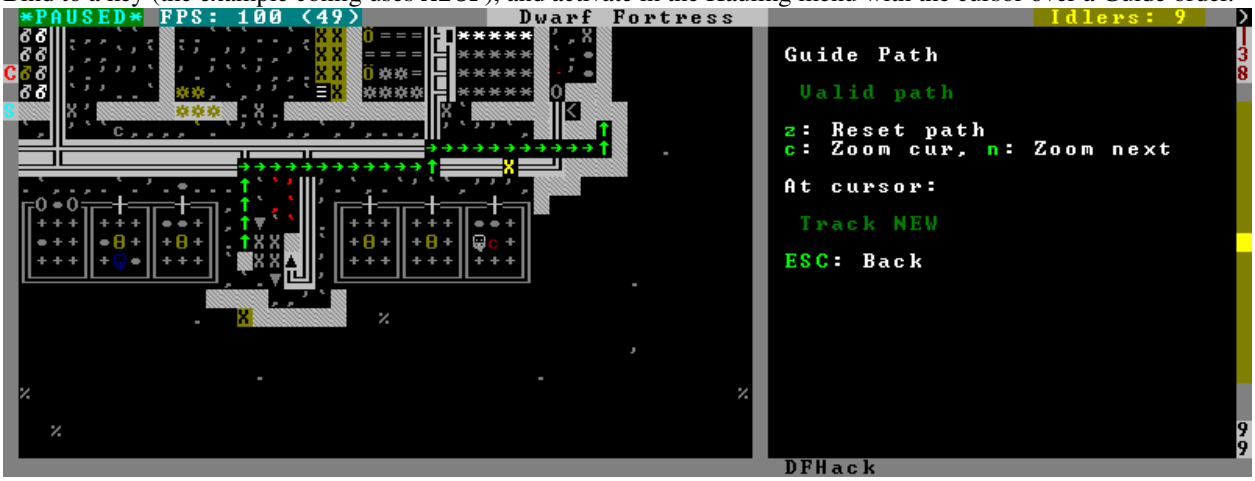


gui/gm-unit

An editor for various unit attributes.

gui/guide-path

Bind to a key (the example config uses `AltP`), and activate in the Hauling menu with the cursor over a Guide order.



The script displays the cached path that will be used by the order; the game computes it when the order is executed for the first time.

gui/hack-wish

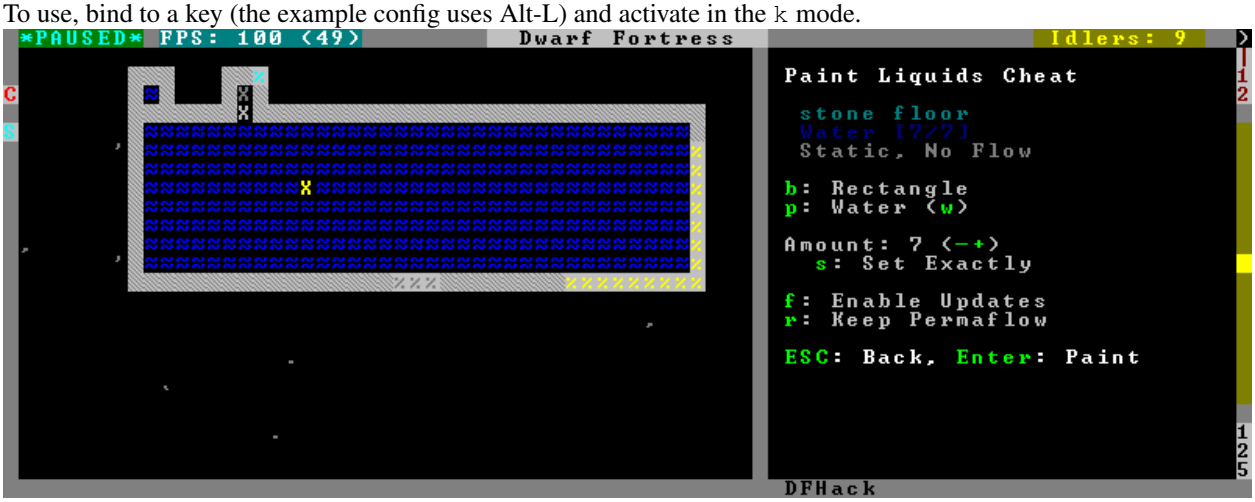
An alias for *gui/create-item*. Deprecated.

gui/hello-world

A basic example for testing, or to start your own script from.

gui/liquids

To use, bind to a key (the example config uses Alt-L) and activate in the `k` mode.



This script is a gui front-end to *liquids* and works similarly, allowing you to add or remove water & magma, and create obsidian walls & floors.

> **Warning:** There is **no undo support**. Bugs in this plugin have been known to create pathfinding problems and heat traps.

The `b` key changes how the affected area is selected. The default *Rectangle* mode works by selecting two corners like any ordinary designation. The `p` key chooses between adding water, magma, obsidian walls & floors, or just tweaking flags.

When painting liquids, it is possible to select the desired level with `+-`, and choose between setting it exactly, only increasing or only decreasing with `s`.

In addition, `f` allows disabling or enabling the flowing water computations for an area, and `r` operates on the "permanent flow" property that makes rivers power water wheels even when full and technically not flowing.

After setting up the desired operations using the described keys, use `Enter` to apply them.

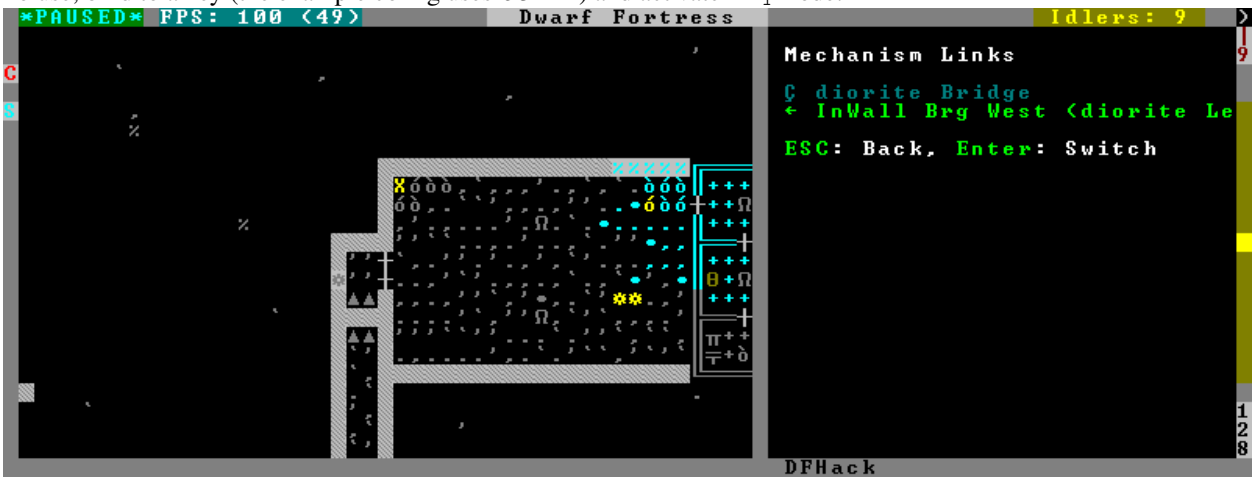
gui/load-screen

A replacement for the "continue game" screen.

Usage: `gui/load-screen enable|disable`

gui/mechanisms

To use, bind to a key (the example config uses `CtrlM`) and activate in `q` mode.



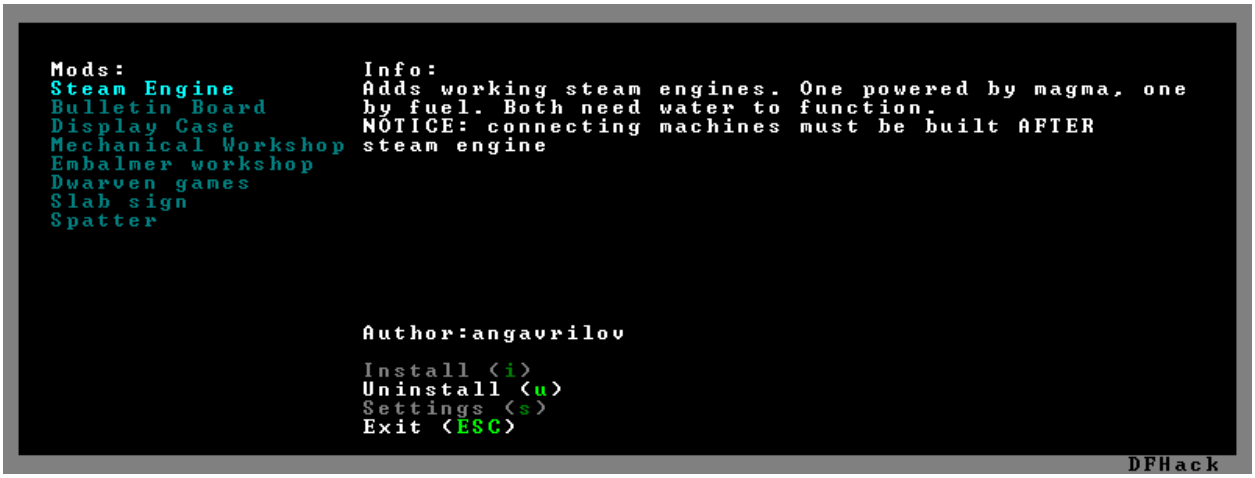
Lists mechanisms connected to the building, and their links. Navigating the list centers the view on the relevant linked buildings.

To exit, press `Esc` or `Enter`; `Esc` recenters on the original building, while `Enter` leaves focus on the current one. `ShiftEnter` has an effect equivalent to pressing `Enter`, and then re-entering the mechanisms UI.

gui/mod-manager

A simple way to install and remove small mods.

It looks for specially formatted mods in `<DF>/mods/`. Mods are not included, but some examples are available here.



gui/no-dfhack-init

Shows a warning at startup if no valid `dfhack.init` file is found.

gui/power-meter

An in-game interface for *power-meter*.

Bind it to a key (default `CtrlShiftM`) and activate after selecting Pressure Plate in the build menu.

The script follows the general look and feel of the regular pressure plate build configuration page, but configures parameters relevant to the modded power meter building.

gui/prerelease-warning

Shows a warning on world load for pre-release builds.

With no arguments passed, the warning is shown unless the "do not show again" option has been selected. With the `force` argument, the warning is always shown.

gui/quickcmd

A list of commands which you can edit while in-game, and which you can execute quickly and easily. For stuff you use often enough to not want to type it, but not often enough to be bothered to find a free keybinding.

gui/rename

Backed by *rename*, this script allows entering the desired name via a simple dialog in the game ui.

- `gui/rename [building]` in `q` mode changes the name of a building.

  The selected building must be one of stockpile, workshop, furnace, trap, or siege engine. It is also possible to rename zones from the `i` menu.

- `gui/rename [unit]` with a unit selected changes the nickname.

  Unlike the built-in interface, this works even on enemies and animals.

- `gui/rename unit-profession` changes the selected unit's custom profession name.

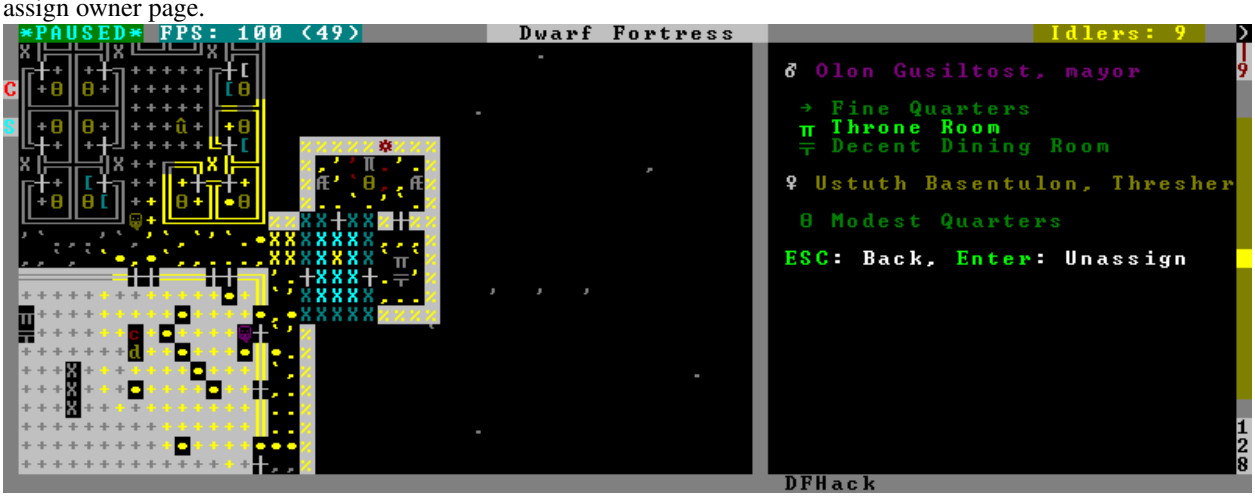
  Likewise, this can be applied to any unit, and when used on animals it overrides their species string.

The `building` or `unit` options are automatically assumed when in relevant UI state.

The example config binds building/unit rename to `CtrlShiftN`, and unit profession change to `CtrlShiftT`.

gui/room-list

To use, bind to a key (the example config uses `AltR`) and activate in `q` mode, either immediately or after opening the assign owner page.



The script lists other rooms owned by the same owner, or by the unit selected in the assign list, and allows unassigning them.

gui/settings-manager

An in-game settings manager (init.txt/d_init.txt)

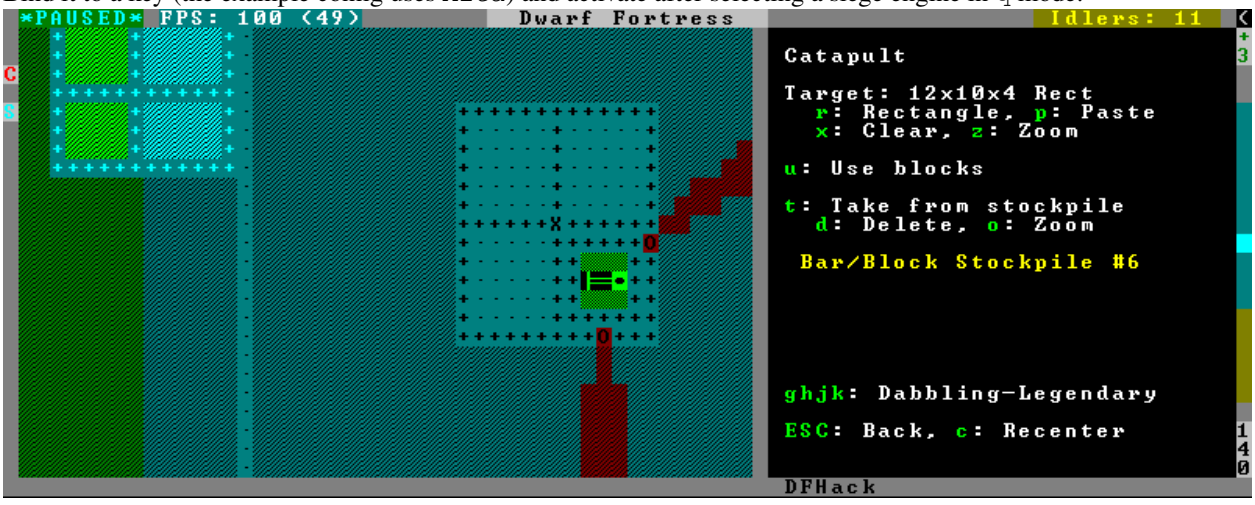
Recommended for use as a keybinding:

```
keybinding add Alt-S@title settings-manager
keybinding add Alt-S@dwarfmode/Default settings-manager
```

gui/siege-engine

An in-game interface for *siege-engine*.

Bind it to a key (the example config uses `Alta`) and activate after selecting a siege engine in `q` mode.



The main mode displays the current target, selected ammo item type, linked stockpiles and the allowed operator skill range. The map tile color is changed to signify if it can be hit by the selected engine: green for fully reachable, blue for out of range, red for blocked, yellow for partially blocked.

Pressing `r` changes into the target selection mode, which works by highlighting two points with `Enter` like all designations. When a target area is set, the engine projectiles are aimed at that area, or units within it (this doesn't actually change the original aiming code, instead the projectile trajectory parameters are rewritten as soon as it appears).

After setting the target in this way for one engine, you can 'paste' the same area into others just by pressing `p` in the main page of this script. The area to paste is kept until you quit DF, or select another area manually.

Pressing `t` switches to a mode for selecting a stockpile to take ammo from.

Exiting from the siege engine script via `Esc` reverts the view to the state prior to starting the script. `ShiftEsc` retains the current viewport, and also exits from the `q` mode to main menu.

gui/stockpiles

An in-game interface for *stocksettings*, to load and save stockpile settings from the `q` menu.

Usage:

gui/stockpiles -save to save the current stockpile

gui/stockpiles -load to load settings into the current stockpile

gui/stockpiles -dir <path> set the default directory to save settings into

gui/stockpiles -help to see this message

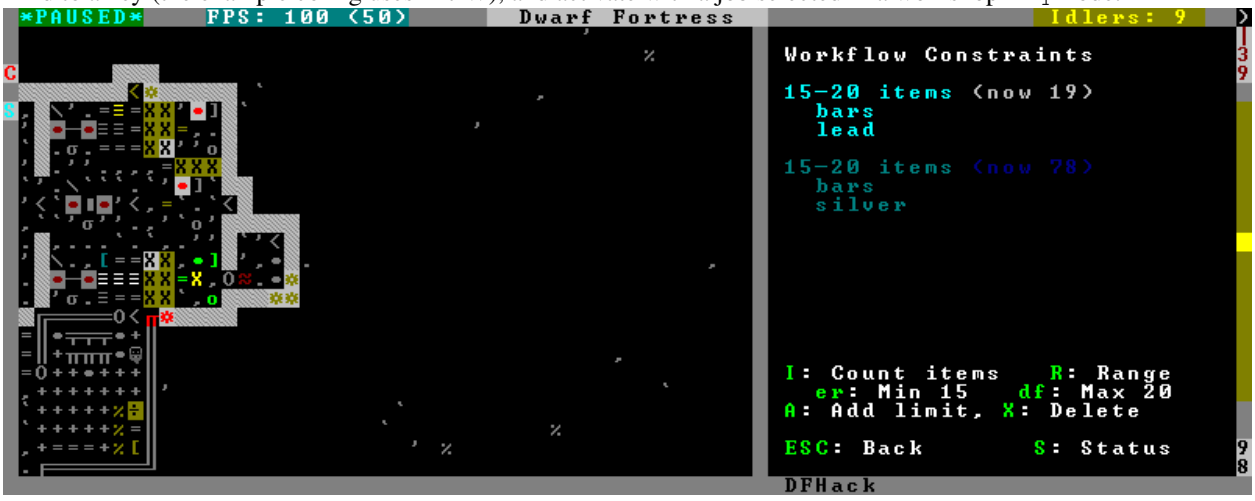
Don't forget to `enable stockpiles` and create the `stocksettings` directory in the DF folder before trying to use the GUI.

gui/unit-info-viewer

Displays age, birth, maxage, shearing, milking, grazing, egg laying, body size, and death info about a unit. Recommended keybinding `AltI`.

gui/workflow

Bind to a key (the example config uses Alt-W), and activate with a job selected in a workshop in `q` mode.



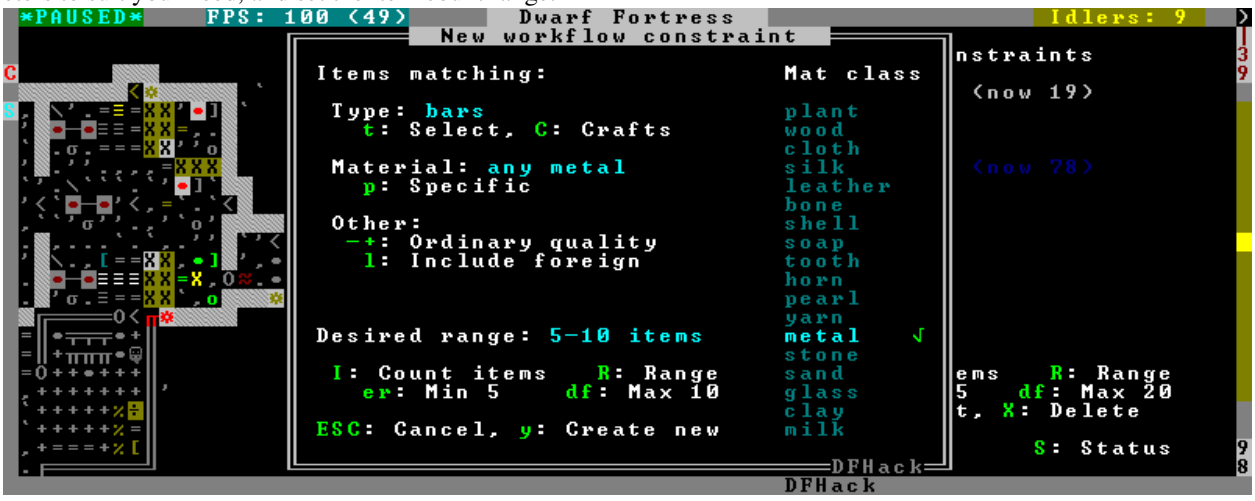
This script provides a simple interface to constraints managed by *workflow*. When active, it displays a list of all constraints applicable to the current job, and their current status.

A constraint specifies a certain range to be compared against either individual *item* or whole *stack* count, an item type and optionally a material. When the current count is below the lower bound of the range, the job is resumed; if it is above or equal to the top bound, it will be suspended. Within the range, the specific constraint has no effect on the job; others may still affect it.

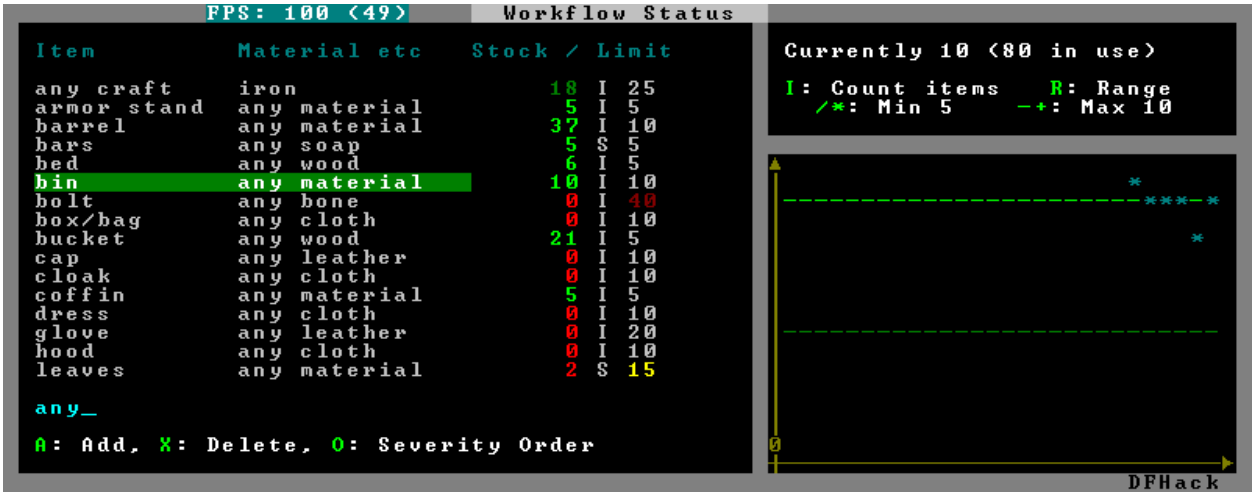
Pressing `i` switches the current constraint between counting stacks or items. Pressing `r` lets you input the range directly; `e`, `r`, `d`, `f` adjust the bounds by 5, 10, or 20 depending on the direction and the `i` setting (counting items and expanding the range each gives a 2x bonus).

Pressing `a` produces a list of possible outputs of this job as guessed by workflow, and lets you create a new constraint by choosing one as template. If you don't see the choice you want in the list, it likely means you have to adjust the job material first using *job* `item-material` or *gui/workshop-job*, as described in the *workflow* documentation. In this manner, this feature can be used for troubleshooting jobs that don't match the right constraints.

If you select one of the outputs with `Enter`, the matching constraint is simply added to the list. If you use `ShiftEnter`, the interface proceeds to the next dialog, which allows you to edit the suggested constraint parameters to suit your need, and set the item count range.



Pressing `s` (or, with the example config, Alt-W in the `z` stocks screen) opens the overall status screen:



This screen shows all currently existing workflow constraints, and allows monitoring and/or changing them from one screen. The constraint list can be filtered by typing text in the field below.

The color of the stock level number indicates how "healthy" the stock level is, based on current count and trend. Bright green is very good, green is good, red is bad, bright red is very bad.

The limit number is also color-coded. Red means that there are currently no workshops producing that item (i.e. no jobs). If it's yellow, that means the production has been delayed, possibly due to lack of input materials.

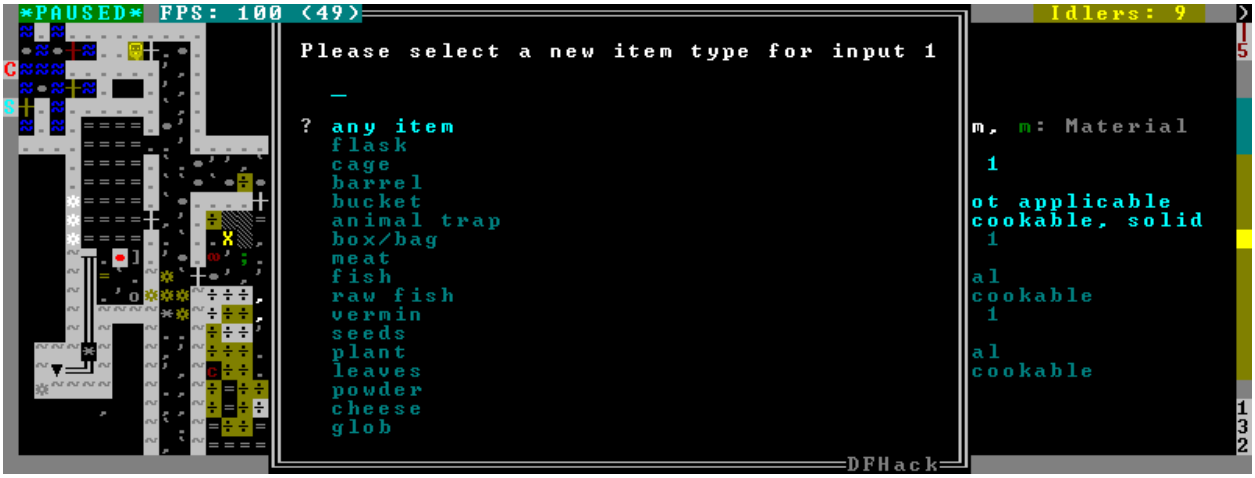
The chart on the right is a plot of the last 14 days (28 half day plots) worth of stock history for the selected item, with the rightmost point representing the current stock value. The bright green dashed line is the target limit (maximum) and the dark green line is that minus the gap (minimum).

gui/workshop-job

Bind to a key (the example config uses `Alta`), and activate with a job selected in a workshop in the `q` mode.

The script shows a list of the input reagents of the selected job, and allows changing them like the *job* `item-type` and *job* `item-material` commands.

Specifically, pressing the `i` key pops up a dialog that lets you select an item type from a list.



Pressing `m`, unless the item type does not allow a material, lets you choose a material.

Since there are a lot more materials than item types, this dialog is more complex and uses a hierarchy of sub-menus. List choices that open a sub-menu are marked with an arrow on the left.

Warning: Due to the way input reagent matching works in DF, you must select an item type if you select a material, or the material will be matched incorrectly in some cases. If you press `m` without choosing an item type, the script will auto-choose if there is only one valid choice, or pop up an error message box instead of the material selection dialog.

Note that both materials and item types presented in the dialogs are filtered by the job input flags, and even the selected item type for material selection, or material for item type selection. Many jobs would let you select only one input item type.

For example, if you choose a *plant* input item type for your prepare meal job, it will only let you select cookable materials.

If you choose a *barrel* item instead (meaning things stored in barrels, like drink or milk), it will let you select any material, since in this case the material is matched against the barrel itself. Then, if you select, say, iron, and then try to change the input item type, now it won't let you select *plant*; you have to unset the material first.

2.4.5 Scripts for Modders

`modtools/*` scripts provide tools for modders, often with changes to the raw files, and are not intended to be called manually by end-users.

These scripts are mostly useful for raw modders and scripters. They all have standard arguments: arguments are of the form `tool -argName1 argVal1 -argName2 argVal2`. This is equivalent to `tool -argName2 argVal2 -argName1 argVal1`. It is not necessary to provide a value to an argument name: `tool -argName3` is fine. Supplying the same argument name multiple times will result in an error. Argument names are preceded with a dash. The `-help` argument will print a descriptive usage string describing the nature of the arguments. For multiple word argument values, brackets must be used: `tool -argName4 [ sadf1 sadf2 sadf3 ]`. In order to allow passing literal braces as part of the argument, backslashes are used: `tool -argName4 [ \] asdf \foo ]` sets `argName4` to `\] asdf foo`. The `*-trigger` scripts have a similar policy with backslashes.

Any of these scripts can be called with `-help` for more information.

Contents

modtools/add-syndrome

This allows adding and removing syndromes from units.

modtools/anonymous-script

This allows running a short simple Lua script passed as an argument instead of running a script from a file. This is useful when you want to do something too complicated to make with the existing modtools, but too simple to be worth its own script file. Example:

```
anonymous-script "print(args[1])" arg1 arg2
# prints "arg1"
```

modtools/create-item

Replaces the *createitem* plugin, with standard arguments. The other versions will be phased out in a later version.

modtools/create-unit

Creates a unit. Use `modtools/create-unit -help` for more info.

modtools/equip-item

Force a unit to equip an item with a particular body part; useful in conjunction with the `create` scripts above. See also *forceequip*.

modtools/extra-gamelog

This script writes extra information to the gamelog. This is useful for tools like Soundsense.

modtools/force

This tool triggers events like megabeasts, caravans, and migrants.

modtools/interaction-trigger

This triggers events when a unit uses an interaction on another. It works by scanning the announcements for the correct attack verb, so the attack verb must be specified in the interaction. It includes an option to suppress this announcement after it finds it.

modtools/invader-item-destroyer

This tool configurably destroys invader items to prevent clutter or to prevent the player from getting tools exclusive to certain races.

modtools/item-trigger

This powerful tool triggers DFHack commands when a unit equips, unequips, or attacks another unit with specified item types, specified item materials, or specified item contaminants.

modtools/moddable-gods

This is a standardized version of Putnam's moddableGods script. It allows you to create gods on the command-line.

modtools/outside-only

This allows you to specify certain custom buildings as outside only, or inside only. If the player attempts to build a building in an inappropriate location, the building will be destroyed.

modtools/projectile-trigger

This triggers dfhack commands when projectiles hit their targets.

modtools/random-trigger

This triggers random dfhack commands with specified probabilities. Register a few scripts, then tell it to "go" and it will pick one based on the probability weights you specified. Outcomes are mutually exclusive. To make independent random events, call the script multiple times.

modtools/raw-lint

Checks for simple issues with raw files. Can be run automatically.

modtools/reaction-product-trigger

This triggers dfhack commands when reaction products are produced, once per product.

modtools/reaction-trigger

Triggers dfhack commands when custom reactions complete, regardless of whether it produced anything, once per completion. Use the `-help` command for more information.

modtools/reaction-trigger-transition

Scans raw files and creates a file to help modders transition from autoSyndrome to reaction-trigger.

Prints useful things to the console and a file to help modders transition from autoSyndrome to reaction-trigger. This script is basically an apology for breaking backward compatibiility, and will be removed eventually.

modtools/skill-change

Sets or modifies a skill of a unit. Args:

-help print the help message

-skill skillName set the skill that we're talking about

-mode (add/set) are we adding experience/levels or setting them?

-granularity (experience/level) direct experience, or experience levels?

-unit id id of the target unit

-value amount how much to set/add

modtools/spawn-flow

Creates flows at the specified location.

modtools/syndrome-trigger

Triggers dfhack commands when syndromes are applied to units.

modtools/transform-unit

Transforms a unit into another unit type, possibly permanently. Warning: this will crash arena mode if you view the unit on the same tick that it transforms. If you wait until later, it will be fine.

CHAPTER 3

Other Contents

3.1 List of Authors

The following is a list of people who have contributed to DFHack, in alphabetical order.

If you should be here and aren't, please get in touch on IRC or the forums, or make a pull request!

Name	Github	Other
8Z	8Z	
acwatkins	acwatkins	
Alexander Gavrilov	angavrilov	ag
AndreasPK	AndreasPK	
Angus Mezick	amezick	
Antalia	tamarakorr	
Anuradha Dissanayake	falconne	
belal	jimhester	
Ben Lubar	BenLubar	
Caldfir	caldfir	
Chris Dombroski	cdombroski	
Clayton Hughes		
David Corbett	dscorbett	
Deon		
DoctorVanGogh	DoctorVanGogh	
Donald Ruegsegger	hashaash	
doomchild	doomchild	
enjia2000		
Eric Wald	eswald	
Erik Youngren	Artanis	
Espen Wiborg		
expwnent	expwnent	
Feng		
Harlan Playford	playfordh	
IndigoFenix		
James Logsdon	jlogsdon	
Japa	JapaMala	
Jared Adams		
Jim Lisi	stonetoad	
jj	jjyg	jj‘‘

Continued on next page

Table 3.1 – continued from previous page

Name	Github	Other
John Beisley	huin	
John Shade	gsvslto	
Jonas Ask		
kane-t	kane-t	
Kelly Martin	ab9rf	
Kris Parker	kaypy	
Kurik Amudnil		
Lethosor	lethosor	
Mason11987	Mason11987	
Matthew Cline		
Max	maxthyme	Max^TM
melkor217	melkor217	
Meneth32		
Meph		
Michon van Dooren	MaienM	
miffedmap	miffedmap	
Mike Stewart	thewonderidiot	
Mikko Juola	Noeda	Adeon
MithrilTuxedo	MithrilTuxedo	
mizipzor	mizipzor	
moversti	moversti	
Neil Little	nmlittle	
Nick Rart	nickrart	comestible
Omniclasm		
PeridexisErrant	PeridexisErrant	
Petr Mrázek	peterix	
potato		
Priit Laes	plaes	
Putnam	Putnam3145	
Quietust	quietust	_Q
Raidau	Raidau	
Ramblurr	Ramblurr	
rampaging-poet		
Raoul van Putten		
Raoul XQ	raoulxq	
reverb		
Rinin	Rinin	
Robert Heinrich	rh73	
Robert Janetzko	robertjanetzko	
rofl0r	rofl0r	
root		
Roses	Pheosics	
Ross M	RossM	
rout		
Rumrusher	rumrusher	
RusAnon	RusAnon	
sami		
scamtank	scamtank	
Sebastian Wolfertz	Enkrod	

Continued on next page

Table 3.1 – continued from previous page

Name	Github	Other
Seth Woodworth	sethwoodworth	
simon		
Simon Jackson	sizeak	
sv-esk	sv-esk	
Tacomagic		
Tim Walberg	twalberg	
Timothy Collett	danaris	
Tom Jobbins	TheBloke	
Tom Prince		
Travis Hoppe	thoppe	orthographic-pedant
txtsd	txtsd	
U-glouglou\simon		
Valentin Ochs	Cat-Ion	
Vjek		
Warmist	warmist	
Wes Malone	wesQ3	
Will Rogers	wjrogers	
Zhentar	Zhentar	
zilpin	zilpin	

3.2 Licenses

DFHack is distributed under the Zlib license, with some MIT- and BSD-licensed components. These licenses protect your right to use DFhack for any purpose, distribute copies, and so on.

The core, plugins, scripts, and other DFHack code all use the ZLib license unless noted otherwise. By contributing to DFHack, authors release the contributed work under this license.

DFHack also draws on several external packages. Their licenses are summarised here and reproduced below.

Component	License	Copyright
DFHack	Zlib	(c) 2009-2012, Petr Mrázek
clsocket	BSD 3-clause	(c) 2007-2009, CarrierLabs, LLC.
dirent	MIT	(c) 2006, Toni Ronkko
JSON.lua	CC-BY-SA	(c) 2010-2014, Jeffrey Friedl
jsoncpp	MIT	(c) 2007-2010, Baptiste Lepilleur
linenoise	BSD 2-clause	(c) 2010, Salvatore Sanfilippo & Pieter Noordhuis
lua	MIT	(c) 1994-2008, Lua.org, PUC-Rio.
luafilesystem	MIT	(c) 2003-2014, Kepler Project
protobuf	BSD 3-clause	(c) 2008, Google Inc.
tinythread	Zlib	(c) 2010, Marcus Geelnard
tinyxml	Zlib	(c) 2000-2006, Lee Thomason
UTF-8-decoder	MIT	(c) 2008-2010, Bjoern Hoehrmann

3.2.1 Zlib License

See https://en.wikipedia.org/wiki/Zlib_License

```
This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any
```

```
damages arising from the use of this software.

Permission is granted to anyone to use this software for any
purpose, including commercial applications, and to alter it and
redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must
   not claim that you wrote the original software. If you use this
   software in a product, an acknowledgment in the product
   documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and
   must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source
   distribution.
```

3.2.2 MIT License

See https://en.wikipedia.org/wiki/MIT_License

```
Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

3.2.3 BSD Licenses

See https://en.wikipedia.org/wiki/BSD_licenses

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in
    the documentation and/or other materials provided with the
    distribution.
```

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

`linenoise` adds no further clauses.

`protobuf` adds the following clause:

```
3. Neither the name of Google Inc. nor the names of its
   contributors may be used to endorse or promote products derived
   from this software without specific prior written permission.
```

`clsocket` adds the following clauses:

```
3. The name of the author may not be used to endorse or promote
   products derived from this software without specific prior
   written permission.

4. The name "CarrierLabs" must not be used to endorse or promote
   products derived from this software without prior written
   permission. For written permission, please contact
   mark@carrierlabs.com
```

3.3 Changelog

Contents

3.3.1 DFHack future

Internals

- Commands to run on startup can be specified on the command line with +

  Example:

```
./dfhack +devel/print-args example
"Dwarf Fortress.exe" +devel/print-args example
```

- Prevented plugins with active viewscreens from being unloaded and causing a crash
- Additional script search paths can be specified in dfhack-config/script-paths.txt

Lua

- *Building-hacks* now supports `auto_gears` flags. It automatically finds and animates gears in building definition
- Changed how *Eventful* triggers reaction complete. Now it has `onReactionComplete` and `onReactionCompleting`. Second one can be canceled

New Plugins

- *autogems*: Creates a new Workshop Order setting, automatically cutting rough gems

New Scripts

- *devel/save-version*: Displays DF version information about the current save
- *modtools/extra-gamelog*: replaces `log-region`, `soundsense-season`, and `soundsense`

New Features

- *buildingplan*: Support for floodgates, grates, and bars
- *colonies*: new `place` subcommand and supports any vermin (default honey bees)
- *confirm*: Added a confirmation for retiring locations
- *exportlegends*: Exports more information (poetic/musical/dance forms, written/artifact content, landmasses, extra histfig information, and more)
- *search*: Support for new screens:
  - location occupation assignment
  - civilization animal training knowledge
  - animal trainer assignment
- *tweak*:
  - `tweak block-labors`: Prevents labors that can't be used from being toggled
  - `tweak hide-priority`: Adds an option to hide designation priority indicators
  - `tweak title-start-rename`: Adds a safe rename option to the title screen "Start Playing" menu
- *zone*:
  - Added `unassign` subcommand
  - Added `only` option to `assign` subcommand

Fixes

- Fixed a crash bug caused by the historical figures DFHack uses to store persistent data.
- More plugins should recognize non-dwarf citizens
- Fixed a possible crash from cloning jobs
- moveToBuilding() now sets flags for items that aren't a structural part of the building properly
- *autotrade*, *stocks*: Made trading work when multiple caravans are present but only some can trade
- *confirm* note-delete: No longer interferes with name entry
- *exportlegends*: Handles entities without specific races, and a few other fixes for things new to v0.42
- *fastdwarf*: Fixed a bug involving teleporting mothers but not the babies they're holding.
- *gaydar*: Fixed text display on OS X/Linux and failure with soul-less creatures
- *manipulator*:
  - allowed editing of non-dwarf citizens
  - stopped ghosts and visitors from being editable
  - fixed applying last custom profession
- *modtools/create-unit*: Stopped making units without civs historical figures
- *modtools/force*:
  - Removed siege option
  - Prevented a crash resulting from a bad civilization option
- *showmood*: Fixed name display on OS X/Linux
- *view-item-info*: Fixed density units

Misc Improvements

- *autochop*: Can now edit log minimum/maximum directly and remove limit entirely
- *autolabor*, *autohauler*, *manipulator*: Added support for new jobs/labors/skills
- *colonies*: now implemented by a script
- *createitem*: Can now create items anywhere without specifying a unit, as long as a unit exists on the map
- *devel/export-dt-ini*: Updated for 0.42.06
- *devel/find-offsets*: Automated several more scans
- *gui/gm-editor*: Now supports finding some items with a numeric ID (with `i`)
- *lua*: Now supports some built-in variables like *gui/gm-editor*, e.g. `unit`, `screen`
- *remotefortressreader*: Can now trigger keyboard events
- *stockflow*: Now offers better control over individual craft jobs
- *weather*: now implemented by a script
- *zone*: colored output

Removed

- DFusion: legacy script system, obsolete or replaced by better alternatives

3.3.2 DFHack 0.40.24-r5

New Features

- *confirm*:
  - Added a `uniform-delete` option for military uniform deletion
  - Added a basic in-game configuration UI

Fixes

- Fixed a rare crash that could result from running *keybinding* in onLoadWorld.init
- Script help that doesn't start with a space is now recognized correctly
- *confirm*: Fixed issues with haul-delete, route-delete, and squad-disband confirmations intercepting keys too aggressively
- *emigration* should work now
- *fix-unit-occupancy*: Significantly optimized - up to 2,000 times faster in large fortresses
- *gui/create-item*: Allow exiting quantity prompt
- *gui/family-affairs*: Fixed an issue where lack of relationships wasn't recognized and other issues
- *modtools/create-unit*: Fixed a possible issue in reclaim fortress mode
- *search*: Fixed a crash on the military screen
- *tweak* max-wheelbarrow: Fixed a minor display issue with large numbers
- *workflow*: Fixed a crash related to job postings (and added a fix for existing, broken jobs)

Misc Improvements

- Unrecognized command feedback now includes more information about plugins
- *fix/dry-buckets*: replaces the `drybuckets` plugin
- *feature*: now implemented by a script

3.3.3 DFHack 0.40.24-r4

Internals

- A method for caching screen output is now available to Lua (and C++)
- Developer plugins can be ignored on startup by setting the `DFHACK_NO_DEV_PLUGINS` environment variable
- The console on Linux and OS X now recognizes keyboard input between prompts
- JSON libraries available (C++ and Lua)

- More DFHack build information used in plugin version checks and available to plugins and lua scripts
- Fixed a rare overflow issue that could cause crashes on Linux and OS X
- Stopped DF window from receiving input when unfocused on OS X
- Fixed issues with keybindings involving `CtrlA` and `CtrlZ`, as well as `AltE`/`U`/`N` on OS X
- Multiple contexts can now be specified when adding keybindings
- Keybindings can now use `F10-F12` and `0-9`
- Plugin system is no longer restricted to plugins that exist on startup
- `dfhack.init` file locations significantly generalized

Lua

- Scripts can be enabled with the built-in *enable*/*disable* commands
- A new function, `reqscript()`, is available as a safer alternative to `script_environment()`
- Lua viewscreens can choose not to intercept the OPTIONS keybinding

New internal commands

- *kill-lua*: Interrupt running Lua scripts
- *type*: Show where a command is implemented

New plugins

- *confirm*: Adds confirmation dialogs for several potentially dangerous actions
- *fix-unit-occupancy*: Fixes issues with unit occupancy, such as faulty "unit blocking tile" messages (Bug 3499)
- *title-version* (formerly `vshook`): Display DFHack version on title screen

New scripts

- *armoks-blessing*: Adjust all attributes, personality, age and skills of all dwarves in play
- *brainwash*: brainwash a dwarf (modifying their personality)
- *burial*: sets all unowned coffins to allow burial ("-pets" to allow pets too)
- *deterioratecloths*: make worn clothes on the ground wear far faster to boost FPS
- *deterioratecorpses*: make body parts wear away far faster to boost FPS
- *deterioratefood*: make food vanish after a few months if not used
- *elevate-mental*: elevate all the mental attributes of a unit
- *elevate-physical*: elevate all the physical attributes of a unit
- *emigration*: stressed dwarves may leave your fortress if they see a chance
- *fix-ster*: changes fertility/sterility of animals or dwarves
- *gui/family-affairs*: investigate and alter romantic relationships
- *make-legendary*: modify skill(s) of a single unit

- *modtools/create-unit*: create new units from nothing
- *modtools/equip-item*: a script to equip items on units
- *points*: set number of points available at embark screen
- *pref-adjust*: Adjust all preferences of all dwarves in play
- *rejuvenate*: make any "old" dwarf 20 years old
- *starvingdead*: make undead weaken after one month on the map, and crumble after six
- *view-item-info*: adds information and customisable descriptions to item viewscreens
- *warn-starving*: check for starving, thirsty, or very drowsy units and pause with warning if any are found

New tweaks

- embark-profile-name: Allows the use of lowercase letters when saving embark profiles
- kitchen-keys: Fixes DF kitchen meal keybindings
- kitchen-prefs-color: Changes color of enabled items to green in kitchen preferences
- kitchen-prefs-empty: Fixes a layout issue with empty kitchen tabs

Fixes

- Plugins with vmethod hooks can now be reloaded on OS X
- Lua's `os.system()` now works on OS X
- Fixed default arguments in Lua gametype detection functions
- Circular lua dependencies (reqscript/script_environment) fixed
- Prevented crash in `Items::createItem()`
- *buildingplan*: Now supports hatch covers
- *gui/create-item*: fixed assigning quality to items, made `Esc` work properly
- *gui/gm-editor*: handles lua tables properly
- *help*: now recognizes built-in commands, like `help`
- *manipulator*: fixed crash when selecting custom professions when none are found
- *remotefortressreader*: fixed crash when attempting to send map info when no map was loaded
- *search*: fixed crash in unit list after cancelling a job; fixed crash when disabling stockpile category after searching in a subcategory
- *stocksettings*: now checks/sanitizes filenames when saving
- *stocks*: fixed a crash when right-clicking
- *steam-engine*: fixed a crash on arena load; number keys (e.g. 2/8) take priority over cursor keys when applicable
- tweak fps-min fixed
- tweak farm-plot-select: Stopped controls from appearing when plots weren't fully built
- *workflow*: Fixed some issues with stuck jobs. Existing stuck jobs must be cancelled and re-added
- *zone*: Fixed a crash when using `zone set` (and a few other potential crashes)

Misc Improvements

- DFHack documentation:
  - massively reorganised, into files of more readable size
  - added many missing entries
  - indexes, internal links, offline search all documents
  - includes documentation of linked projects (df-structures, third-party scripts)
  - better HTML generation with Sphinx
  - documentation for scripts now located in source files
- *autolabor*:
  - Stopped modification of labors that shouldn't be modified for brokers/diplomats
  - Prioritize skilled dwarves more efficiently
  - Prevent dwarves from running away with tools from previous jobs
- *automaterial*: Fixed several issues with constructions being allowed/disallowed incorrectly when using box-select
- *dwarfmonitor*:
  - widgets' positions, formats, etc. are now customizable
  - weather display now separated from the date display
  - New mouse cursor widget
- *gui/dfstatus*: Can enable/disable individual categories and customize metal bar list
- *full-heal*: `-r` option removes corpses
- *gui/gm-editor*
  - Pointers can now be displaced
  - Added some useful aliases: "item" for the selected item, "screen" for the current screen, etc.
  - Now avoids errors with unrecognized types
- *gui/hack-wish*: renamed to *gui/create-item*
- *keybinding list* accepts a context
- *lever*:
  - Lists lever names
  - `lever pull` can be used to pull the currently-selected lever
- `memview`: Fixed display issue
- *modtools/create-item*: arguments are named more clearly, and you can specify the creator to be the unit with id `df.global.unit_next_id-1` (useful in conjunction with *modtools/create-unit*)
- `nyan`: Can now be stopped with dfhack-run
- *plug*: lists all plugins; shows state and number of commands in plugins
- *prospect*: works from within command-prompt
- *quicksave*: Restricted to fortress mode

- *remotefortressreader*: Exposes more information
- *search*:
  - Supports noble suggestion screen (e.g. suggesting a baron)
  - Supports fortress mode loo[k] menu
  - Recognizes ? and ; keys
- *stocks*: can now match beginning and end of item names
- *teleport*: Fixed cursor recognition
- *tidlers*, *twaterlvl*: now implemented by scripts instead of a plugin
- *tweak*:
  - debug output now logged to stderr.log instead of console - makes DFHack start faster
  - farm-plot-select: Fixed issues with selecting undiscovered crops
- *workflow*: Improved handling of plant reactions

Removed

- *embark-tools* nano: 1x1 embarks are now possible in vanilla 0.40.24

3.3.4 DFHack 0.40.24-r3

Internals

- Ruby library now included on OS X - Ruby scripts should work on OS X 10.10
- libstdc++ should work with older versions of OS X
- Added support for *onMapLoad.init / onMapUnload.init* scripts
- game type detection functions are now available in the World module
- The `DFHACK_LOG_MEM_RANGES` environment variable can be used to log information to `stderr.log` on OS X
- Fixed adventure mode menu names
- Fixed command usage information for some commands

Lua

- Lua scripts will only be reloaded if necessary
- Added a `df2console()` wrapper, useful for printing DF (CP437-encoded) text to the console in a portable way
- Added a `strerror()` wrapper

New Internal Commands

- *hide*, *show*: hide and show the console on Windows
- *sc-script*: Allows additional scripts to be run when certain events occur (similar to *onLoad*.init* scripts)

New Plugins

- *autohauler*: A hauling-only version of autolabor

New Scripts

- *modtools/reaction-product-trigger*: triggers callbacks when products are produced (contrast with when reactions complete)

New Tweaks

- *fps-min*: Fixes the in-game minimum FPS setting
- *shift-8-scroll*: Gives Shift+8 (or *) priority when scrolling menus, instead of scrolling the map
- *tradereq-pet-gender*: Displays pet genders on the trade request screen

Fixes

- Fixed game type detection in *3dveins*, *gui/create-item*, *reveal*, *seedwatch*
- `PRELOAD_LIB`: More extensible on Linux
- *add-spatter*, *Eventful*: Fixed crash on world load
- *add-thought*: Now has a proper subthought arg.
- *Building-hacks*: Made buildings produce/consume correct amount of power
- *fix-armory*: compiles and is available again (albeit with issues)
- *gui/gm-editor*: Added search option (accessible with "s")
- *hack-wish*: Made items stack properly.
- *modtools/skill-change*: Made level granularity work properly.
- *show-unit-syndromes*: should work
- *stockflow*:
  - Fixed error message in Arena mode
  - no longer checks the DF version
  - fixed ballistic arrow head orders
  - convinces the bookkeeper to update records more often
- *zone*: Stopped crash when scrolling cage owner list

Misc Improvements

- *autolabor*: A negative pool size can be specified to use the most unskilled dwarves
- *Building-hacks*:
  - Added a way to allow building to work even if it consumes more power than is available.
  - Added setPower/getPower functions.
- *catsplosion*: Can now trigger pregnancies in (most) other creatures

- *exportlegends*: `info` and `all` options export `legends_plus.xml` with more data for legends utilities
- *manipulator*:
  - Added ability to edit nicknames/profession names
  - added "Job" as a View Type, in addition to "Profession" and "Squad"
  - added custom profession templates with masking
- *remotefortressreader*: Exposes more information

3.3.5 DFHack 0.40.24-r2

Internals

- Lua scripts can set environment variables of each other with `dfhack.run_script_with_env`
- Lua scripts can now call each others internal nonlocal functions with `dfhack.script_environment(scriptName).functionName(arg1,arg2)`
- *Eventful*: Lua reactions no longer require LUA_HOOK as a prefix; you can register a callback for the completion of any reaction with a name
- Filesystem module now provides file access/modification times and can list directories (normally and recursively)
- Units Module: New functions:

```
isWar
isHunter
isAvailableForAdoption
isOwnCiv
isOwnRace
getRaceName
getRaceNamePlural
getRaceBabyName
getRaceChildName
isBaby
isChild
isAdult
isEggLayer
isGrazer
isMilkable
isTrainableWar
isTrainableHunting
isTamable
isMale
isFemale
isMerchant
isForest
isMarkedForSlaughter
```

- Buildings Module: New Functions:

```
isActivityZone
isPenPasture
isPitPond
isActive
findPenPitAt
```

Fixes

- `dfhack.run_script` should correctly find save-specific scripts now.
- *add-thought*: updated to properly affect stress.
- *hfs-pit*: should work now
- *autobutcher*: takes gelding into account
- `init.lua` existence checks should be more reliable (notably when using non-English locales)

Misc Improvements

Multiline commands are now possible inside dfhack.init scripts. See `dfhack.init-example` for example usage.

3.3.6 DFHack 0.40.24-r1

Internals

CMake shouldn't cache DFHACK_RELEASE anymore. People may need to manually update/delete their CMake cache files to get rid of it.

3.3.7 DFHack 0.40.24-r0

Internals

- *Events from EventManager*: fixed crash error with EQUIPMENT_CHANGE event.
- key modifier state exposed to Lua (ie `Ctrl`, `Alt`, `Shift`)

Fixes

`dfhack.sh` can now be run from other directories on OS X

New Plugins

- *blueprint*: export part of your fortress to quickfort .csv files

New Scripts

- *hotkey-notes*: print key, name, and jump position of hotkeys

Removed

- needs_porting/*

Misc Improvements

Added support for searching more lists

3.3.8 Older Changelogs

Are kept in a seperate file: HISTORY

CHAPTER 4

For Developers

4.1 How to contribute to DFHack

Contents

4.1.1 Contributing Code

Several things should be kept in mind when contributing code to DFHack.

Code Format

- Four space indents for C++. Never use tabs for indentation in any language.
- LF (Unix style) line terminators
- Avoid trailing whitespace
- UTF-8 encoding
- For C++:
  - Opening and closing braces on their own lines or opening brace at the end of the previous line
  - Braces placed at original indent level if on their own lines
  - #includes should be sorted. C++ libraries first, then dfhack modules, then df structures, then local includes. Within each category they should be sorted alphabetically.

How to get new code into DFHack

- Submit pull requests to the `develop` branch, not the `master` branch. (The `master` branch always points at the most recent release)
- Use a new branch for each feature or bugfix so that your changes can be merged independently (i.e. not the master or develop branch of your fork).
- If possible, compile on multiple platforms when changing anything that compiles
- It must pass CI - run `python travis/all.py` to check this.
- Update `NEWS.rst` and `docs/Authors.rst` when applicable.
- Create a GitHub pull request once finished
- Submit ideas and bug reports as issues on GitHub. Posts in the forum thread can easily get missed or forgotten.
- Work on reported problems will take priority over ideas or suggestions.

Memory research

If you want to do memory research, you'll need some tools and some knowledge. In general, you'll need a good memory viewer and optionally something to look at machine code without getting crazy :) Using publicly known information and analyzing the game's data is preferred.

Good windows tools include:

- Cheat Engine
- IDA Pro 5.0 (freely available for non-commercial use)

Good linux tools:

- angavrilov's df-structures gui (visit us on IRC for details).
- edb (Evan's Debugger)
- IDA Pro 5.0 running under Wine
- Some of the tools residing in the `legacy` dfhack branch.

4.1.2 Using the library as a developer

Currently, the most direct way to use the library is to write a script or plugin that can be loaded by it. All the plugins can be found in the 'plugins' folder. There's no in-depth documentation on how to write one yet, but it should be easy enough to copy one and just follow the pattern. `plugins/skeleton/skeleton.cpp` is provided for this purpose.

Other than through plugins, it is possible to use DFHack via remote access interface, or by writing scripts in Lua or Ruby. There are plenty of examples in the scripts folder. The *DFHack Lua API* is quite well documented.

The most important parts of DFHack are the Core, Console, Modules and Plugins.

- Core acts as the centerpiece of DFHack - it acts as a filter between DF and SDL and synchronizes the various plugins with DF.
- Console is a thread-safe console that can be used to invoke commands exported by Plugins.
- Modules actually describe the way to access information in DF's memory. You can get them from the Core. Most modules are split into two parts: high-level and low-level. High-level is mostly method calls, low-level publicly visible pointers to DF's data structures.

- Plugins are the tools that use all the other stuff to make things happen. A plugin can have a list of commands that it exports and an onupdate function that will be called each DF game tick.

Rudimentary API documentation can be built using doxygen (see build options in `CMakeCache.txt` or with `ccmake` or `cmake-gui`). The full DFHack documentation is built with Sphinx, which runs automatically at compile time.

DFHack consists of variously licensed code, but invariably weak copyleft. The main license is zlib/libpng, some bits are MIT licensed, and some are BSD licensed. See the *Licenses* for more information.

Feel free to add your own extensions and plugins. Contributing back to the DFHack repository is welcome and the right thing to do :)

DF data structure definitions

DFHack uses information about the game data structures, represented via xml files in the `library/xml/` submodule.

See https://github.com/DFHack/df-structures, and the documentation linked in the index.

Data structure layouts are described in files following the `df.*.xml` name pattern. This information is transformed by a perl script into C++ headers describing the structures, and associated metadata for the Lua wrapper. These headers and data are then compiled into the DFHack libraries, thus necessitating a compatibility break every time layouts change; in return it significantly boosts the efficiency and capabilities of DFHack code.

Global object addresses are stored in `symbols.xml`, which is copied to the dfhack release package and loaded as data at runtime.

Remote access interface

DFHack supports remote access by exchanging Google protobuf messages via a TCP socket. Both the core and plugins can define remotely accessible methods. The `dfhack-run` command uses this interface to invoke ordinary console commands.

Currently the supported set of requests is limited, because the developers don't know what exactly is most useful. *remotefortressreader* provides a fairly comprehensive interface for visualisers such as Armok Vision.

4.1.3 Documentation Standards

DFHack documentation is built with Sphinx, and configured automatically through CMake. If you want to build the docs *only*, use this command:

```
sphinx-build . docs/html
```

Whether you're adding new code or just fixing old documentation (and there's plenty), there are a few important standards for completeness and consistent style. Treat this section as a guide rather than iron law, match the surrounding text, and you'll be fine.

Each command should have a short (~54 character) help string, which is shown by the *ls* command. For scripts, this is a comment on the first line (the comment marker and whitespace is stripped). For plugins it's the second argument to `PluginCommand`. Please make this brief but descriptive!

Everything should be documented! If it's not clear *where* a particular thing should be documented, ask on IRC or in the DFHack thread on Bay12 - as well as getting help, you'll be providing valuable feedback that makes it easier for future readers!

Scripts can use a custom autodoc function, based on the Sphinx `include` directive and Ruby docstring conventions - any lines between `=begin` and `=end` are copied into the appropriate scripts documentation page. They **must** have a heading which exactly matches the command, underlined with `=====` to the same length. For example, a lua file would have:

```
--[[=begin

add-thought
===========
Adds a thought or emotion to the selected unit.  Can be used by other scripts,
or the gui invoked by running ``add-thought gui`` with a unit selected.

=end]]
```

Ruby scripts use the same syntax, but obviously omit the leading `--[[` and trailing `]]` which denote a multiline comment in lua. `=begin` and `=end` are native syntax (and matched in lua for convenience).

Where the heading for a section is also the name of a command, the spelling and case should exactly match the command to enter in the DFHack command line.

Try to keep lines within 80-100 characters, so it's readable in plain text - Sphinx (our documentation system) will make sure paragraphs flow.

If there aren't many options or examples to show, they can go in a paragraph of text. Use double-backticks to put commands in monospaced font, like this:

```
You can use ``cleanowned scattered x`` to dump tattered or abandoned items.
```

If the command takes more than three arguments, format the list as a table called Usage. The table *only* lists arguments, not full commands. Input values are specified in angle brackets. Example:

```
Usage:

:arg1:          A simple argument.
:arg2 <input>:  Does something based on the input value.
:Very long argument:
                Is very specific.
```

To demonstrate usage - useful mainly when the syntax is complicated, list the full command with arguments in monospaced font, then indent the next line and describe the effect:

```
``resume all``
        Resumes all suspended constructions.
```

If it would be helpful to mention another DFHack command, don't just type the name - add a hyperlink! Specify the link target in backticks, and it will be replaced with the corresponding title and linked: eg `` `autolabor` => *autolabor*``. Link targets should be equivalent to the command described (without file extension), and placed above the heading of that section like this:

```
.. _autolabor:

autolabor
=========
```

Add link targets if you need them, but otherwise plain headings are preferred. Scripts using the in-source docs option, which should be all of them, have link targets created automatically.

4.1.4 Other ways to help

DFHack is a software project, but there's a lot more to it than programming. If you're not comfortable programming, you can help by:

- reporting bugs and incomplete documentation
- improving the documentation
- finding third-party scripts to add
- writing tutorials for newbies

All those things are crucial, and often under-represented. So if that's your thing, go get started!

4.2 Compiling DFHack

You don't need to compile DFHack unless you're developing plugins or working on the core.

For users, modders, and authors of scripts it's better to download and *install the latest release instead*.

Contents

4.2.1 How to get the code

DFHack doesn't have any kind of system of code snapshots in place, so you will have to get code from the GitHub repository using Git. How to get Git is described under the instructions for each platform.

To get the latest release code (master branch):

```
git clone --recursive https://github.com/DFHack/dfhack
cd dfhack
```

If your version of Git does not support the `--recursive` flag, you will need to omit it and run `git submodule update --init` after entering the dfhack directory.

To get the latest development code (develop branch), clone as above and then:

```
git checkout develop
git submodule update
```

Important note regarding submodule update and changing branches:

You must run `git submodule update` every time you change Git branch, for example when switching between master and develop branches and back.

4.2.2 Contributing to DFHack

If you want to get involved with the development, create an account on GitHub, make a clone there and then use that as your remote repository instead.

We'd love that; join us on IRC (#dfhack channel on freenode) for discussion, and whenever you need help.

For lots more details on contributing to DFHack, including pull requests, code format, and more, please see *Contributing Code*.

4.2.3 Build types

`cmake` allows you to pick a build type by changing the `CMAKE_BUILD_TYPE` variable:

```
cmake .. -DCMAKE_BUILD_TYPE:string=BUILD_TYPE
```

Without specifying a build type or 'None', cmake uses the `CMAKE_CXX_FLAGS` variable for building.

Valid and useful build types include 'Release', 'Debug' and 'RelWithDebInfo'. 'Debug' is not available on Windows, use 'RelWithDebInfo' instead.

4.2.4 Linux

On Linux, DFHack acts as a library that shadows parts of the SDL API using LD_PRELOAD.

Dependencies

DFHack is meant to be installed into an existing DF folder, so get one ready.

We assume that any Linux platform will have `git` available (though it may require installing from your package manager.)

To build DFHack you need GCC version 4.5 or later, capable of compiling for 32-bit (i386) targets. GCC 4.5 is easiest to work with due to avoiding libstdc++ issues (see below), but any version from 4.5 onwards (including 5.x) will work.

On 64-bit distributions, you'll need the multilib development tools and libraries:

- `gcc-multilib` and `g++-multilib`
- If you have installed a non-default version of GCC - for example, GCC 4.5 on a distribution that defaults to 5.x - you may need to add the version number to the multilib packages.
  - For example, `gcc-4.5-multilib` and `g++-4.5-multilib` if installing for GCC 4.5 on a system that uses a later GCC version.
  - This is definitely required on Ubuntu/Debian, check if using a different distribution.

Note that installing a 32-bit GCC on 64-bit systems (e.g. `gcc:i386` on Debian) will typically *not* work, as it depends on several other 32-bit libraries that conflict with system libraries. Alternatively, you might be able to use `lxc` to create a virtual 32-bit environment.

Before you can build anything, you'll also need `cmake`. It is advisable to also get `ccmake` on distributions that split the cmake package into multiple parts.

You also need perl and the XML::LibXML and XML::LibXSLT perl packages (for the code generation parts). You should be able to find them in your distro repositories.

To build stonesense, you'll also need OpenGL headers.

Here are some package install commands for various platforms:

- On Arch linux:
  - For the required Perl modules: `perl-xml-libxml` and `perl-xml-libxslt` (or through `cpan`)
- On 64-bit Ubuntu:

```
apt-get install gcc cmake git gcc-multilib g++-multilib zlib1g-dev:i386 libxml-libxml-perl libxm
```

- On 32-bit Ubuntu:

```
apt-get install gcc cmake git gcc-multilib g++-multilib zlib1g-dev libxml-libxml-perl libxml-lib
```

- Debian and derived distros should have similar requirements to Ubuntu.

Build

Building is fairly straightforward. Enter the `build` folder (or create an empty folder in the DFHack directory to use instead) and start the build like this:

```
cd build
cmake .. -DCMAKE_BUILD_TYPE:string=Release -DCMAKE_INSTALL_PREFIX=<path to DF>
make install # or make -jX install on multi-core systems to compile with X parallel processes
```

<path to DF> should be a path to a copy of Dwarf Fortress, of the appropriate version for the DFHack you are building. This will build the library along with the normal set of plugins and install them into your DF folder.

Alternatively, you can use ccmake instead of cmake:

```
cd build
ccmake ..
make install
```

This will show a curses-based interface that lets you set all of the extra options. You can also use a cmake-friendly IDE like KDevelop 4 or the cmake-gui program.

Incompatible libstdc++

When compiling dfhack yourself, it builds against your system libstdc++. When Dwarf Fortress runs, it uses a libstdc++ shipped with the binary, which comes from GCC 4.5 and is incompatible with code compiled with newer GCC versions. This manifests itself with an error message such as:

```
./libs/Dwarf_Fortress: /pathToDF/libs/libstdc++.so.6: version
    `GLIBCXX_3.4.15' not found (required by ./hack/libdfhack.so)
```

To fix this you can compile with GCC 4.5 or remove the libstdc++ shipped with DF, which causes DF to use your system libstdc++ instead:

```
cd /path/to/DF/
rm libs/libstdc++.so.6
```

Note that distributing binaries compiled with newer GCC versions requires end- users to delete libstdc++ themselves and have a libstdc++ on their system from the same GCC version or newer. For this reason, distributing anything compiled with GCC versions newer than 4.5 is discouraged. In the future we may start bundling a later libstdc++ as part of the DFHack package, so as to enable compilation-for-distribution with a GCC newer than 4.5.

4.2.5 Mac OS X

DFHack functions similarly on OS X and Linux, and the majority of the information above regarding the build process (cmake and make) applies here as well.

If you have issues building on OS X 10.10 (Yosemite) or above, try definining the following environment variable:

```
export MACOSX_DEPLOYMENT_TARGET=10.9
```

Note for El Capitan (OSX 10.11) and XCode 7.x users

- You will probably find when following the instructions below that GCC 4.5 will fail to install on OSX 10.11, or any older OSX that is using XCode 7.
- There are two workarounds:
  - Install GCC 5.x instead (`brew install gcc5`), and then after compile replace `hack/libstdc++.6.dylib` with a symlink to GCC 5's i386 version of this file:

```
cd <path to df>/hack && mv libstdc++.6.dylib libstdc++.6.dylib.orig &&
ln -s /usr/local/Cellar/gcc5/5.2.0/lib/gcc/5/i386/libstdc++.6.dylib .
```

  - Install XCode 6, which is available as a free download from the Apple Developer Center.
    - Either install this as your only XCode, or install it additionally to XCode 7 and then switch between them using `xcode-select`
    - Ensure XCode 6 is active before attempting to install GCC 4.5 and whenever you are compiling DFHack with GCC 4.5.

Dependencies and system set-up

1. Download and unpack a copy of the latest DF
2. Install Xcode from Mac App Store
3. Install the XCode Command Line Tools by running the following command:

```
xcode-select --install
```

4. Install dependencies

   Using Homebrew (recommended):

```
brew tap homebrew/versions
brew install git
brew install cmake
brew install gcc45
```

   Using MacPorts:

```
sudo port install gcc45 +universal cmake +universal git-core +universal
```

   Macports will take some time - maybe hours. At some point it may ask you to install a Java environment; let it do so.

   It is recommended to use Homebrew instead of MacPorts, as it is generally cleaner, quicker, and smarter. For example, installing MacPort's GCC 4.5 will install more than twice as many dependencies as Homebrew's will, and all in both 32bit and 64bit variants. Homebrew also doesn't require constant use of sudo.

5. Install Perl dependencies

- Using system Perl
  - `sudo cpan`

    If this is the first time you've run cpan, you will need to go through the setup process. Just stick with the defaults for everything and you'll be fine.

    If you are running OS X 10.6 (Snow Leopard) or earlier, good luck! You'll need to open a separate Terminal window and run:

```
sudo ln -s /usr/include/libxml2/libxml /usr/include/libxml
```

  - `install XML::LibXML`
  - `install XML::LibXSLT`
- In a separate, local Perl install

  Rather than using system Perl, you might also want to consider the Perl manager, Perlbrew.

  This manages Perl 5 locally under `~/perl5/`, providing an easy way to install Perl and run CPAN against it without `sudo`. It can maintain multiple Perl installs and being local has the benefit of easy migration and insulation from OS issues and upgrades.

  See http://perlbrew.pl/ for more details.

Building

- Get the DFHack source as per section *How to get the code*, above.
- Set environment variables

  Homebrew (if installed elsewhere, replace /usr/local with `$(brew --prefix)`):

```
export CC=/usr/local/bin/gcc-4.5
export CXX=/usr/local/bin/g++-4.5
```

  Macports:

```
export CC=/opt/local/bin/gcc-mp-4.5
export CXX=/opt/local/bin/g++-mp-4.5
```

  Change the version numbers appropriately if you installed a different version of GCC.
- Build dfhack:

```
mkdir build-osx
cd build-osx
cmake .. -DCMAKE_BUILD_TYPE:string=Release -DCMAKE_INSTALL_PREFIX=<path to DF>
make install # or make -j X install on multi-core systems to compile with X parallel processes
```

  <path to DF> should be a path to a copy of Dwarf Fortress, of the appropriate version for the DFHack you are building.

4.2.6 Windows

On Windows, DFHack replaces the SDL library distributed with DF.

Dependencies

You will need the following:

- Microsoft Visual Studio 2010 SP1, with the C++ language
- Git
- CMake
- Perl with XML::LibXML and XML::LibXSLT
  - It is recommended to install StrawberryPerl, which includes both.

Microsoft Visual Studio 2010 SP1

DFHack has to be compiled with the Microsoft Visual C++ 2010 SP1 toolchain; later versions won't work against Dwarf Fortress due to ABI and STL incompatibilities.

At present, the only way to obtain the MSVC C++ 2010 toolchain is to install a full copy of Microsoft Visual Studio 2010 SP1. The free Express version is sufficient.

You can grab it from Microsoft's site.

You should also install the Visual Studio 2010 SP1 update.

You can confirm whether you have SP1 by opening the Visual Studio 2010 IDE and selecting About from the Help menu. If you have SP1 it will have *SP1Rel* at the end of the version number, for example: *Version 10.0.40219.1 SP1Rel*

Use of pre-SP1 releases has been reported to cause issues and is therefore not supported by DFHack. Please ensure you are using SP1 before raising any Issues.

If your Windows Update is configured to receive updates for all Microsoft Products, not just Windows, you will receive the SP1 update automatically through Windows Update (you will probably need to trigger a manual check.)

If not, you can download it directly from this Microsoft Download link.

Additional dependencies: installing with the Chocolatey Package Manager

The remainder of dependencies - Git, CMake and StrawberryPerl - can be most easily installed using the Chocolatey Package Manger. Chocolatey is a *nix-style package manager for Windows. It's fast, small (8-20MB on disk) and very capable. Think "`apt-get` for Windows."

Chocolatey is a preferred way of installing the required dependencies as it's quicker, less effort and will install known-good utilities guaranteed to have the correct setup (especially PATH).

To install Chocolatey and the required dependencies:

- Go to https://chocolatey.org in a web browser
- At the top of the page it will give you the install command to copy
  - Copy the first one, which starts `@powershell ...`
  - It won't be repeated here in case it changes in future Chocolatey releases.
- Open an elevated (Admin) `cmd.exe` window
  - On Windows 8 and later this can be easily achieved by:
    - right-clicking on the Start Menu, or pressing Win+X.
    - choosing "Command Prompt (Admin)"

  - On earlier Windows: find `cmd.exe` in Start Menu, right click and choose Open As Administrator.
- Paste in the Chocolatey install command and hit enter
- Close this `cmd.exe` window and open another Admin `cmd.exe` in the same way
- Run the following command:

```
choco install git cmake.portable strawberryperl -y
```

- Close the Admin `cmd.exe` window; you're done!

You can now use all of these utilities from any normal `cmd.exe` window. You only need Admin/elevated `cmd.exe` for running `choco install` commands; for all other purposes, including compiling DFHack, you should use a normal `cmd.exe` (or, better, an improved terminal like Cmder; details below, under Build.)

NOTE: you can run the above `choco install` command even if you already have Git, CMake or StrawberryPerl installed. Chocolatey will inform you if any software is already installed and won't re-install it. In that case, please check the PATHs are correct for that utility as listed in the manual instructions below. Or, better, manually uninstall the version you have already and re-install via Chocolatey, which will ensure the PATH are set up right and will allow Chocolatey to manage that program for you in future.

Additional dependencies: installing manually

If you prefer to install manually rather than using Chocolatey, details and requirements are as below. If you do install manually, please ensure you have all PATHs set up correctly.

Git Some examples:

- Git for Windows (command-line and GUI)
- tortoisegit (GUI and File Explorer integration)

CMake You can get the win32 installer version from the official site. It has the usual installer wizard. Make sure you let it add its binary folder to your binary search PATH so the tool can be later run from anywhere.

Perl / Strawberry Perl For the code generation parts you'll need Perl 5 with XML::LibXML and XML::LibXSLT. Strawberry Perl is recommended as it includes all of the required packages in a single, easy install.

After install, ensure Perl is in your user's PATH. This can be edited from `Control Panel -> System -> Advanced System Settings -> Environment Variables`.

The following three directories must be in PATH, in this order:

- `<path to perl>\c\bin`
- `<path to perl>\perl\site\bin`
- `<path to perl>\perl\bin`

Be sure to close and re-open any existing `cmd.exe` windows after updating your PATH.

If you already have a different version of Perl (for example the one from Cygwin), you can run into some trouble. Either remove the other Perl install from PATH, or install XML::LibXML and XML::LibXSLT for it using CPAN.

Build

There are several different batch files in the `build` folder along with a script that's used for picking the DF path.

First, run `set_df_path.vbs` and point the dialog that pops up at a suitable DF installation which is of the appropriate version for the DFHack you are compiling. The result is the creation of the file `DF_PATH.txt` in the build directory. It contains the full path to the destination directory. You could therefore also create this file manually - or copy in a pre-prepared version - if you prefer.

Next, run one of the scripts with `generate` prefix. These create the MSVC solution file(s):

- `all` will create a solution with everything enabled (and the kitchen sink).
- `gui` will pop up the CMake GUI and let you choose what to build. This is probably what you want most of the time. Set the options you are interested in, then hit configure, then generate. More options can appear after the configure step.
- `minimal` will create a minimal solution with just the bare necessities - the main library and standard plugins.

Then you can either open the solution with MSVC or use one of the msbuild scripts:

Building/installing from the command line:

In the build directory you will find several `.bat` files:

- Scripts with `build` prefix will only build DFHack.
- Scripts with `install` prefix will build DFHack and install it to the previously selected DF path.
- Scripts with `package` prefix will build and create a .zip package of DFHack.

Compiling from the command line is generally the quickest and easiest option. However be aware that due to the limitations of `cmd.exe` - especially in versions of Windows prior to Windows 10 - it can be very hard to see what happens during a build. If you get a failure, you may miss important errors or warnings due to the tiny window size and extremely limited scrollback. For that reason you may prefer to compile in the IDE which will always show all build output.

Alternatively (or additionally), consider installing an improved Windows terminal such as Cmder. Easily installed through Chocolatey with: `choco install cmder -y`.

Note for Cygwin/msysgit users: It is also possible to compile DFHack from a Bash command line. This has three potential benefits:

- When you've installed Git and are using its Bash, but haven't added Git to your path:
  - You can load Git's Bash and as long as it can access Perl and CMake, you can use it for compile without adding Git to your system path.
- When you've installed Cygwin and its SSH server:
  - You can now SSH in to your Windows install and compile from a remote terminal; very useful if your Windows installation is a local VM on a *nix host OS.
- In general: you can use Bash as your compilation terminal, meaning you have a decent sized window, scrollback, etc.
  - Whether you're accessing it locally as with Git's Bash, or remotely through Cygwin's SSH server, this is far superior to using `cmd.exe`.

You don't need to do anything special to compile from Bash. As long as your PATHs are set up correctly, you can run the same generate- and build/install/package- bat files as detailed above.

Building/installing from the Visual Studio IDE:

After running the CMake generate script you will have a new folder called VC2010. Open the file `dfhack.sln` inside that folder. If you have multiple versions of Visual Studio installed, make sure you open with Visual Studio 2010.

The first thing you must then do is change the build type. It defaults to Debug, but this cannot be used on Windows. Debug is not binary-compatible with DF. If you try to use a debug build with DF, you'll only get crashes and for this reason the Windows "debug" scripts actually do RelWithDebInfo builds. After loading the Solution, change the Build Type to either `Release` or `RelWithDebInfo`.

Then build the `INSTALL` target listed under `CMakePredefinedTargets`.

4.2.7 Building the documentation

DFHack documentation, like the file you are reading now, is created as .rst files, which are in reStructuredText (reST) format. This is a documenation format that has come from the Python community. It is very similar in concept - and in syntax - to Markdown, as found on GitHub and many other places. However it is more advanced than Markdown, with more features available when compiled to HTML, such as automatic tables of contents, cross-linking, special external links (forum, wiki, etc) and more. The documentation is compiled by a Python tool, Sphinx.

The DFHack build process will compile the documentation but this has been disabled by default. You only need to build the docs if you're changing them, or perhaps if you want a local HTML copy; otherwise, read them easily online at ReadTheDoc's DFHack hosted documentation.

(Note that even if you do want a local copy, it is certainly not necesesary to compile the documentation in order to read it. Like Markdown, reST documents are designed to be just as readable in a plain-text editor as they are in HTML format. The main thing you lose in plain text format is hyperlinking.)

Enabling documentation building

First, make sure you have followed all the necessary steps for your platform as outlined in the rest of this document.

To compile documentation with DFHack, add the following flag to your `cmake` command:

```
-DBUILD_DOCS:bool=ON
```

For example:

```
cmake .. -DCMAKE_BUILD_TYPE:string=Release -DBUILD_DOCS:bool=ON -DCMAKE_INSTALL_PREFIX=<path to DF>
```

Alternatively you can use the CMake GUI which allows options to be changed easily.

On Windows you should either use `generate-msvc-gui.bat` and set the option through the GUI, or else if you want to use an alternate file, such as `generate-msvc-all.bat`, you will need to edit it to add the flag. Or you could just run `cmake` on the command line like in other platforms.

Required dependencies

In order to build the documentation, you must have Python with Sphinx version 1.3.1 or later. Both Python 2.x and 3.x are supported.

When installing Sphinx from OS package managers, be aware that there is another program called Sphinx, completely unrelated to documentation management. Be sure you are installing the right Sphinx; it may be called `python-sphinx`, for example. To avoid doubt, `pip` can be used instead as detailed below.

Linux

Most Linux distributions will include Python as standard.

Check your package manager to see if Sphinx 1.3.1 or later is available, but at the time of writing Ubuntu for example only has 1.2.x.

You can instead install Sphinx with the pip package manager. This may need to be installed from your OS package manager; this is the case on Ubuntu. On Ubuntu/Debian, use the following to first install pip:

```
sudo apt-get install python-pip
```

Once pip is available, you can then install the Python Sphinx module with:

```
pip install sphinx
```

If you run this as a normal user it will install a local copy for your user only. Run it with sudo if you want a system-wide install. Either is fine for DFHack, however if installing locally do check that `sphinx-build` is in your path. It may be installed in a directory such as `~/.local/bin/`, so after pip install, find `sphinx-build` and ensure its directory is in your local `$PATH`.

Mac OS X

OS X has Python 2.7 installed by default, but it does not have the pip package manager.

You can install Homebrew's Python 3, which includes pip, and then install the latest Sphinx using pip:

```
brew install python3
pip3 install sphinx
```

Alternatively, you can simply install Sphinx 1.3.x directly from Homebrew:

```
brew install sphinx-doc
```

This will install Sphinx for OS X's system Python 2.7, without needing pip.

Either method works; if you plan to use Python for other purposes, it might best to install Homebrew's Python 3 so that you have the latest Python as well as pip. If not, just installing sphinx-doc for OS X's system Python 2.7 is fine.

Windows

Use the Chocolatey package manager to install Python and pip, then use pip to install Sphinx.

Run the following commands from an elevated (Admin) `cmd.exe`, after installing Chocolatey as outlined in the *Windows section*:

```
choco install python pip -y
```

Then close that Admin `cmd.exe`, re-open another Admin `cmd.exe`, and run:

```
pip install sphinx
```

4.3 DFHack Lua API

DFHack has extensive support for the Lua scripting language, providing access to:

1. Raw data structures used by the game.

2. Many C++ functions for high-level access to these structures, and interaction with dfhack itself.
3. Some functions exported by C++ plugins.

Lua code can be used both for writing scripts, which are treated by DFHack command line prompt almost as native C++ commands, and invoked by plugins written in C++.

This document describes native API available to Lua in detail. It does not describe all of the utility functions implemented by Lua files located in `hack/lua/*` (`library/lua/*` in the git repo).

Contents

4.3.1 DF data structure wrapper

Data structures of the game are defined in XML files located in `library/xml` (and online, and automatically exported to lua code as a tree of objects and functions under the `df` global, which also broadly maps to the `df` namespace in the headers generated for C++.

> **Warning:** The wrapper provides almost raw access to the memory of the game, so mistakes in manipulating objects are as likely to crash the game as equivalent plain C++ code would be.
> eg. NULL pointer access is safely detected, but dangling pointers aren't.

Objects managed by the wrapper can be broadly classified into the following groups:

1. Typed object pointers (references).

   References represent objects in DF memory with a known type.

   In addition to fields and methods defined by the wrapped type, every reference has some built-in properties and methods.

2. Untyped pointers

   Represented as lightuserdata.

   In assignment to a pointer NULL can be represented either as `nil`, or a NULL lightuserdata; reading a NULL pointer field returns `nil`.

3. Named types

   Objects in the `df` tree that represent identity of struct, class, enum and bitfield types. They host nested named types, static methods, builtin properties & methods, and, for enums and bitfields, the bi-directional mapping between key names and values.

4. The `global` object

   `df.global` corresponds to the `df::global` namespace, and behaves as a mix between a named type and a reference, containing both nested types and fields corresponding to global symbols.

In addition to the `global` object and top-level types the `df` global also contains a few global builtin utility functions.

Typed object references

The underlying primitive lua object is userdata with a metatable. Every structured field access produces a new userdata instance.

All typed objects have the following built-in features:

- `ref1 == ref2`, `tostring(ref)`

  References implement equality by type & pointer value, and string conversion.

- `pairs(ref)`

  Returns an iterator for the sequence of actual C++ field names and values. Fields are enumerated in memory order. Methods and lua wrapper properties are not included in the iteration.

  > **Warning:** a few of the data structures (like ui_look_list) contain unions with pointers to different types with vtables. Using pairs on such structs is an almost sure way to crash with an access violation.

- `ref._kind`

  Returns one of: `primitive`, `struct`, `container`, or `bitfield`, as appropriate for the referenced object.

- `ref._type`

  Returns the named type object or a string that represents the referenced object type.

- `ref:sizeof()`

  Returns *size, address*

- `ref:new()`

  Allocates a new instance of the same type, and copies data from the current object.

- `ref:delete()`

  Destroys the object with the C++ `delete` operator. If destructor is not available, returns *false*.

  Warning: the lua reference object remains as a dangling pointer, like a raw C++ pointer would.

- `ref:assign(object)`

  Assigns data from object to ref. Object must either be another ref of a compatible type, or a lua table; in the latter case special recursive assignment rules are applied.

- `ref:_displace(index[,step])`

  Returns a new reference with the pointer adjusted by index*step. Step defaults to the natural object size.

Primitive references

References of the *_kind* `'primitive'` are used for objects that don't fit any of the other reference types. Such references can only appear as a value of a pointer field, or as a result of calling the `_field()` method.

They behave as structs with one field `value` of the right type.

To make working with numeric buffers easier, they also allow numeric indices. Note that other than excluding negative values no bound checking is performed, since buffer length is not available. Index 0 is equivalent to the `value` field.

Struct references

Struct references are used for class and struct objects.

They implement the following features:

- `ref.field`, `ref.field = value`

  Valid fields of the structure may be accessed by subscript.

  Primitive typed fields, i.e. numbers & strings, are converted to/from matching lua values. The value of a pointer is a reference to the target, or `nil`/NULL. Complex types are represented by a reference to the field within the structure; unless recursive lua table assignment is used, such fields can only be read.

  Note: In case of inheritance, *superclass* fields have precedence over the subclass, but fields shadowed in this way can still be accessed as `ref['subclasstype.field']`.

  This shadowing order is necessary because vtable-based classes are automatically exposed in their exact type, and the reverse rule would make access to superclass fields unreliable.

- `ref._field(field)`

  Returns a reference to a valid field. That is, unlike regular subscript, it returns a reference to the field within the structure even for primitive typed fields and pointers.

- `ref:vmethod(args...)`

  Named virtual methods are also exposed, subject to the same shadowing rules.

- `pairs(ref)`

  Enumerates all real fields (but not methods) in memory order, which is the same as declaration order.

Container references

Containers represent vectors and arrays, possibly resizable.

A container field can associate an enum to the container reference, which allows accessing elements using string keys instead of numerical indices.

Note that two-dimensional arrays in C++ (ie pointers to pointers) are exposed to lua as one-dimensional. The best way to handle this is probably `array[x].value:_displace(y)`.

Implemented features:

- `ref._enum`

  If the container has an associated enum, returns the matching named type object.

- `#ref`

  Returns the *length* of the container.

- `ref[index]`

  Accesses the container element, using either a *0-based* numerical index, or, if an enum is associated, a valid enum key string.

  Accessing an invalid index is an error, but some container types may return a default value, or auto-resize instead for convenience. Currently this relaxed mode is implemented by df-flagarray aka BitArray.

- `ref._field(index)`

  Like with structs, returns a pointer to the array element, if possible. Flag and bit arrays cannot return such pointer, so it fails with an error.

- `pairs(ref)`, `ipairs(ref)`

  If the container has no associated enum, both behave identically, iterating over numerical indices in order. Otherwise, ipairs still uses numbers, while pairs tries to substitute enum keys whenever possible.

- `ref:resize(new_size)`

  Resizes the container if supported, or fails with an error.

- `ref:insert(index,item)`

  Inserts a new item at the specified index. To add at the end, use `#ref`, or just `'#'` as index.

- `ref:erase(index)`

  Removes the element at the given valid index.

Bitfield references

Bitfields behave like special fixed-size containers. Consider them to be something in between structs and fixed-size vectors.

The `_enum` property points to the bitfield type. Numerical indices correspond to the shift value, and if a subfield occupies multiple bits, the `ipairs` order would have a gap.

Since currently there is no API to allocate a bitfield object fully in GC-managed lua heap, consider using the lua table assignment feature outlined below in order to pass bitfield values to dfhack API functions that need them, e.g. `matinfo:matches{metal=true}`.

Named types

Named types are exposed in the `df` tree with names identical to the C++ version, except for the `::` vs `.` difference.

All types and the global object have the following features:

- `type._kind`

  Evaluates to one of `struct-type`, `class-type`, `enum-type`, `bitfield-type` or `global`.

- `type._identity`

  Contains a lightuserdata pointing to the underlying `DFHack::type_instance` object.

Types excluding the global object also support:

- `type:sizeof()`

  Returns the size of an object of the type.

- `type:new()`

  Creates a new instance of an object of the type.

- `type:is_instance(object)`

  Returns true if object is same or subclass type, or a reference to an object of same or subclass type. It is permissible to pass `nil`, NULL or non-wrapper value as object; in this case the method returns `nil`.

In addition to this, enum and bitfield types contain a bi-directional mapping between key strings and values, and also map `_first_item` and `_last_item` to the min and max values.

Struct and class types with instance-vector attribute in the xml have a `type.find(key)` function that wraps the find method provided in C++.

Global functions

The `df` table itself contains the following functions and values:

- `NULL`, `df.NULL`

  Contains the NULL lightuserdata.

- `df.isnull(obj)`

  Evaluates to true if obj is nil or NULL; false otherwise.

- `df.isvalid(obj[,allow_null])`

  For supported objects returns one of `type`, `voidptr`, `ref`.

  If *allow_null* is true, and obj is nil or NULL, returns `null`.

Otherwise returns *nil*.

- `df.sizeof(obj)`

  For types and refs identical to `obj:sizeof()`. For lightuserdata returns *nil, address*

- `df.new(obj)`, `df.delete(obj)`, `df.assign(obj, obj2)`

  Equivalent to using the matching methods of obj.

- `df._displace(obj,index[,step])`

  For refs equivalent to the method, but also works with lightuserdata (step is mandatory then).

- `df.is_instance(type,obj)`

  Equivalent to the method, but also allows a reference as proxy for its type.

- `df.new(ptype[,count])`

  Allocate a new instance, or an array of built-in types. The `ptype` argument is a string from the following list: `string`, `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, `int32_t`, `uint32_t`, `int64_t`, `uint64_t`, `bool`, `float`, `double`. All of these except `string` can be used with the count argument to allocate an array.

- `df.reinterpret_cast(type,ptr)`

  Converts ptr to a ref of specified type. The type may be anything acceptable to `df.is_instance`. Ptr may be *nil*, a ref, a lightuserdata, or a number.

  Returns *nil* if NULL, or a ref.

Recursive table assignment

Recursive assignment is invoked when a lua table is assigned to a C++ object or field, i.e. one of:

- `ref:assign{...}`
- `ref.field = {...}`

The general mode of operation is that all fields of the table are assigned to the fields of the target structure, roughly emulating the following code:

```
function rec_assign(ref,table)
    for key,value in pairs(table) do
        ref[key] = value
    end
end
```

Since assigning a table to a field using = invokes the same process, it is recursive.

There are however some variations to this process depending on the type of the field being assigned to:

1. If the table contains an `assign` field, it is applied first, using the `ref:assign(value)` method. It is never assigned as a usual field.
2. When a table is assigned to a non-NULL pointer field using the `ref.field = {...}` syntax, it is applied to the target of the pointer instead.

   If the pointer is NULL, the table is checked for a `new` field:

   (a) If it is *nil* or *false*, assignment fails with an error.

   (b) If it is *true*, the pointer is initialized with a newly allocated object of the declared target type of the pointer.

   (c) Otherwise, `table.new` must be a named type, or an object of a type compatible with the pointer. The pointer is initialized with the result of calling `table.new:new()`.

After this auto-vivification process, assignment proceeds as if the pointer wasn't NULL.

Obviously, the `new` field inside the table is always skipped during the actual per-field assignment processing.

3. If the target of the assignment is a container, a separate rule set is used:

   (a) If the table contains neither `assign` nor `resize` fields, it is interpreted as an ordinary *1-based* lua array. The container is resized to the #-size of the table, and elements are assigned in numeric order:

```
ref:resize(#table);
for i=1,#table do ref[i-1] = table[i] end
```

   (b) Otherwise, `resize` must be *true*, *false*, or an explicit number. If it is not false, the container is resized. After that the usual struct-like 'pairs' assignment is performed.

   In case `resize` is *true*, the size is computed by scanning the table for the largest numeric key.

This means that in order to reassign only one element of a container using this system, it is necessary to use:

```
{ resize=false, [idx]=value }
```

Since `nil` inside a table is indistinguishable from missing key, it is necessary to use `df.NULL` as a null pointer value.

This system is intended as a way to define a nested object tree using pure lua data structures, and then materialize it in C++ memory in one go. Note that if pointer auto-vivification is used, an error in the middle of the recursive walk would not destroy any objects allocated in this way, so the user should be prepared to catch the error and do the necessary cleanup.

4.3.2 DFHack API

DFHack utility functions are placed in the `dfhack` global tree.

Native utilities

Input & Output

- `dfhack.print(args...)`

  Output tab-separated args as standard lua print would do, but without a newline.

- `print(args...)`, `dfhack.println(args...)`

  A replacement of the standard library print function that works with DFHack output infrastructure.

- `dfhack.printerr(args...)`

  Same as println; intended for errors. Uses red color and logs to stderr.log.

- `dfhack.color([color])`

  Sets the current output color. If color is *nil* or *-1*, resets to default. Returns the previous color value.

- `dfhack.is_interactive()`

  Checks if the thread can access the interactive console and returns *true* or *false*.

- `dfhack.lineedit([prompt[,history_filename]])`

  If the thread owns the interactive console, shows a prompt and returns the entered string. Otherwise returns *nil, error*.

  Depending on the context, this function may actually yield the running coroutine and let the C++ code release the core suspend lock. Using an explicit `dfhack.with_suspend` will prevent this, forcing the function to block on input with lock held.

- `dfhack.interpreter([prompt[,history_filename[,env]]])`

  Starts an interactive lua interpreter, using the specified prompt string, global environment and command-line history file.

  If the interactive console is not accessible, returns *nil, error*.

Exception handling

- `dfhack.error(msg[,level[,verbose]])`

  Throws a dfhack exception object with location and stack trace. The verbose parameter controls whether the trace is printed by default.

- `qerror(msg[,level])`

  Calls `dfhack.error()` with `verbose` being *false*. Intended to be used for user-caused errors in scripts, where stack traces are not desirable.

- `dfhack.pcall(f[,args...])`

  Invokes f via xpcall, using an error function that attaches a stack trace to the error. The same function is used by SafeCall in C++, and dfhack.safecall.

- `safecall(f[,args...])`, `dfhack.safecall(f[,args...])`

  Just like pcall, but also prints the error using printerr before returning. Intended as a convenience function.

- `dfhack.saferesume(coroutine[,args...])`

  Compares to coroutine.resume like dfhack.safecall vs pcall.

- `dfhack.exception`

  Metatable of error objects used by dfhack. The objects have the following properties:

  `err.where` The location prefix string, or *nil*.

  `err.message` The base message string.

  `err.stacktrace` The stack trace string, or *nil*.

  `err.cause` A different exception object, or *nil*.

  `err.thread` The coroutine that has thrown the exception.

  `err.verbose` Boolean, or *nil*; specifies if where and stacktrace should be printed.

  `tostring(err)`, or `err:tostring([verbose])` Converts the exception to string.

- `dfhack.exception.verbose`

  The default value of the `verbose` argument of `err:tostring()`.

Miscellaneous

- `dfhack.VERSION`

  DFHack version string constant.

- `dfhack.curry(func,args...)`, or `curry(func,args...)`

  Returns a closure that invokes the function with args combined both from the curry call and the closure call itself. I.e. `curry(func,a,b)(c,d)` equals `func(a,b,c,d)`.

Locking and finalization

- `dfhack.with_suspend(f[,args...])`

  Calls `f` with arguments after grabbing the DF core suspend lock. Suspending is necessary for accessing a consistent state of DF memory.

  Returned values and errors are propagated through after releasing the lock. It is safe to nest suspends.

  Every thread is allowed only one suspend per DF frame, so it is best to group operations together in one big critical section. A plugin can choose to run all lua code inside a C++-side suspend lock.

- `dfhack.call_with_finalizer(num_cleanup_args,always,cleanup_fn[,cleanup_args...],fn[,ar`

  Invokes `fn` with `args`, and after it returns or throws an error calls `cleanup_fn` with `cleanup_args`. Any return values from `fn` are propagated, and errors are re-thrown.

  The `num_cleanup_args` integer specifies the number of `cleanup_args`, and the `always` boolean specifies if cleanup should be called in any case, or only in case of an error.

- `dfhack.with_finalize(cleanup_fn,fn[,args...])`

  Calls `fn` with arguments, then finalizes with `cleanup_fn`. Implemented using `call_with_finalizer(0,true,...)`.

- `dfhack.with_onerror(cleanup_fn,fn[,args...])`

  Calls `fn` with arguments, then finalizes with `cleanup_fn` on any thrown error. Implemented using `call_with_finalizer(0,false,...)`.

- `dfhack.with_temp_object(obj,fn[,args...])`

  Calls `fn(obj,args...)`, then finalizes with `obj:delete()`.

Persistent configuration storage

This api is intended for storing configuration options in the world itself. It probably should be restricted to data that is world-dependent.

Entries are identified by a string `key`, but it is also possible to manage multiple entries with the same key; their identity is determined by `entry_id`. Every entry has a mutable string `value`, and an array of 7 mutable `int`s.

- `dfhack.persistent.get(key)`, `entry:get()`

  Retrieves a persistent config record with the given string key, or refreshes an already retrieved entry. If there are multiple entries with the same key, it is undefined which one is retrieved by the first version of the call.

  Returns entry, or *nil* if not found.

- `dfhack.persistent.delete(key)`, `entry:delete()`

  Removes an existing entry. Returns *true* if succeeded.

- `dfhack.persistent.get_all(key[,match_prefix])`

  Retrieves all entries with the same key, or starting with key..'/'. Calling `get_all('',true)` will match all entries.

  If none found, returns nil; otherwise returns an array of entries.

- `dfhack.persistent.save({key=str1, ...}[,new])`, `entry:save([new])`

  Saves changes in an entry, or creates a new one. Passing true as new forces creation of a new entry even if one already exists; otherwise the existing one is simply updated. Returns *entry, did_create_new*

Since the data is hidden in data structures owned by the DF world, and automatically stored in the save game, these save and retrieval functions can just copy values in memory without doing any actual I/O. However, currently every entry has a 180+-byte dead-weight overhead.

It is also possible to associate one bit per map tile with an entry, using these two methods:

- `entry:getTilemask(block[, create])`

  Retrieves the tile bitmask associated with this entry in the given map block. If `create` is *true*, an empty mask is created if none exists; otherwise the function returns *nil*, which must be assumed to be the same as an all-zero mask.

- `entry:deleteTilemask(block)`

  Deletes the associated tile mask from the given map block.

Note that these masks are only saved in fortress mode, and also that deleting the persistent entry will **NOT** delete the associated masks.

Material info lookup

A material info record has fields:

- `type`, `index`, `material`

  DF material code pair, and a reference to the material object.

- `mode`

  One of `'builtin'`, `'inorganic'`, `'plant'`, `'creature'`.

- `inorganic`, `plant`, `creature`

  If the material is of the matching type, contains a reference to the raw object.

- `figure`

  For a specific creature material contains a ref to the historical figure.

Functions:

- `dfhack.matinfo.decode(type,index)`

  Looks up material info for the given number pair; if not found, returs *nil*.

- `....decode(matinfo)`, `....decode(item)`, `....decode(obj)`

  Uses `matinfo.type`/`matinfo.index`, item getter vmethods, or `obj.mat_type`/`obj.mat_index` to get the code pair.

- `dfhack.matinfo.find(token[,token...])`

  Looks up material by a token string, or a pre-split string token sequence.

- `dfhack.matinfo.getToken(...)`, `info:getToken()`

  Applies `decode` and constructs a string token.

- `info:toString([temperature[,named]])`

  Returns the human-readable name at the given temperature.

- `info:getCraftClass()`

  Returns the classification used for craft skills.

- `info:matches(obj)`

  Checks if the material matches job_material_category or job_item. Accept dfhack_material_category auto-assign table.

Random number generation

- `dfhack.random.new([seed[,perturb_count]])`

  Creates a new random number generator object. Without any arguments, the object is initialized using current time. Otherwise, the seed must be either a non-negative integer, or a list of such integers. The second argument may specify the number of additional randomization steps performed to improve the initial state.

- `rng:init([seed[,perturb_count]])`

  Re-initializes an already existing random number generator object.

- `rng:random([limit])`

  Returns a random integer. If `limit` is specified, the value is in the range [0, limit); otherwise it uses the whole 32-bit unsigned integer range.

- `rng:drandom()`

  Returns a random floating-point number in the range [0,1).

- `rng:drandom0()`

  Returns a random floating-point number in the range (0,1).

- `rng:drandom1()`

  Returns a random floating-point number in the range [0,1].

- `rng:unitrandom()`

  Returns a random floating-point number in the range [-1,1].

- `rng:unitvector([size])`

  Returns multiple values that form a random vector of length 1, uniformly distributed over the corresponding sphere surface. The default size is 3.

- `fn = rng:perlin([dim]); fn(x[,y[,z]])`

  Returns a closure that computes a classical Perlin noise function of dimension *dim*, initialized from this random generator. Dimension may be 1, 2 or 3 (default).

C++ function wrappers

Thin wrappers around C++ functions, similar to the ones for virtual methods. One notable difference is that these explicit wrappers allow argument count adjustment according to the usual lua rules, so trailing false/nil arguments can be omitted.

- `dfhack.getOSType()`

  Returns the OS type string from `symbols.xml`.

- `dfhack.getDFVersion()`

  Returns the DF version string from `symbols.xml`.

- `dfhack.getDFHackVersion()`
- `dfhack.getDFHackRelease()`
- `dfhack.getCompiledDFVersion()`
- `dfhack.getGitDescription()`
- `dfhack.getGitCommit()`
- `dfhack.isRelease()`

  Return information about the DFHack build in use.

  Note: `getCompiledDFVersion()` returns the DF version specified at compile time, while `getDFVersion()` returns the version and typically the OS as well. These do not necessarily match - for example, DFHack 0.34.11-r5 worked with DF 0.34.10 and 0.34.11, so the former function would always return `0.34.11` while the latter would return `v0.34.10 <platform>` or `v0.34.11 <platform>`.

- `dfhack.getDFPath()`

  Returns the DF directory path.

- `dfhack.getHackPath()`

  Returns the dfhack directory path, i.e. `"../df/hack/"`.

- `dfhack.getSavePath()`

  Returns the path to the current save directory, or *nil* if no save loaded.

- `dfhack.getTickCount()`

  Returns the tick count in ms, exactly as DF ui uses.

- `dfhack.isWorldLoaded()`

  Checks if the world is loaded.

- `dfhack.isMapLoaded()`

  Checks if the world and map are loaded.

- `dfhack.TranslateName(name[,in_english,only_last_name])`

  Convert a language_name or only the last name part to string.

- `dfhack.df2utf(string)`

  Convert a string from DF's CP437 encoding to UTF-8.

- `dfhack.df2console()`

  Convert a string from DF's CP437 encoding to the correct encoding for the DFHack console.

- `dfhack.utf2df(string)`

  Convert a string from UTF-8 to DF's CP437 encoding.

Note: When printing CP437-encoded text to the console (for example, names returned from TranslateName()), use `print(dfhack.df2console(text)` to ensure proper display on all platforms.

Gui module

- `dfhack.gui.getCurViewscreen([skip_dismissed])`

  Returns the topmost viewscreen. If `skip_dismissed` is *true*, ignores screens already marked to be removed.

- `dfhack.gui.getFocusString(viewscreen)`

  Returns a string representation of the current focus position in the ui. The string has a "screen/foo/bar/baz..." format.

- `dfhack.gui.getCurFocus([skip_dismissed])`

  Returns the focus string of the current viewscreen.

- `dfhack.gui.getViewscreenByType(type [, depth])`

  Returns the topmost viewscreen out of the top `depth` viewscreens with the specified type (e.g. `df.viewscreen_titlest`), or `nil` if none match. If `depth` is not specified or is less than 1, all viewscreens are checked.

- `dfhack.gui.getSelectedWorkshopJob([silent])`

  When a job is selected in `q` mode, returns the job, else prints error unless silent and returns *nil*.

- `dfhack.gui.getSelectedJob([silent])`

  Returns the job selected in a workshop or unit/jobs screen.

- `dfhack.gui.getSelectedUnit([silent])`

  Returns the unit selected via `v`, `k`, unit/jobs, or a full-screen item view of a cage or suchlike.

- `dfhack.gui.getSelectedItem([silent])`

  Returns the item selected via `v` ->inventory, `k`, `t`, or a full-screen item view of a container. Note that in the last case, the highlighted *contained item* is returned, not the container itself.

- `dfhack.gui.getSelectedBuilding([silent])`

  Returns the building selected via `q`, `t`, `k` or `i`.

- `dfhack.gui.writeToGamelog(text)`

  Writes a string to `gamelog.txt` without doing an announcement.

- `dfhack.gui.makeAnnouncement(type,flags,pos,text,color[,is_bright])`

  Adds an announcement with given announcement_type, text, color, and brightness. The is_bright boolean actually seems to invert the brightness.

  The announcement is written to `gamelog.txt`. The announcement_flags argument provides a custom set of `announcements.txt` options, which specify if the message should actually be displayed in the announcement list, and whether to recenter or show a popup.

  Returns the index of the new announcement in `df.global.world.status.reports`, or -1.

- `dfhack.gui.addCombatReport(unit,slot,report_index)`

  Adds the report with the given index (returned by makeAnnouncement) to the specified group of the given unit. Returns *true* on success.

- `dfhack.gui.addCombatReportAuto(unit,flags,report_index)`

  Adds the report with the given index to the appropriate group(s) of the given unit, as requested by the flags.

- `dfhack.gui.showAnnouncement(text,color[,is_bright])`

  Adds a regular announcement with given text, color, and brightness. The is_bright boolean actually seems to invert the brightness.

- `dfhack.gui.showZoomAnnouncement(type,pos,text,color[,is_bright])`

  Like above, but also specifies a position you can zoom to from the announcement menu.

- `dfhack.gui.showPopupAnnouncement(text,color[,is_bright])`

  Pops up a titan-style modal announcement window.

- `dfhack.gui.showAutoAnnouncement(type,pos,text,color[,is_bright,unit1,unit2])`

  Uses the type to look up options from announcements.txt, and calls the above operations accordingly. The units are used to call `addCombatReportAuto`.

Job module

- `dfhack.job.cloneJobStruct(job)`

  Creates a deep copy of the given job.

- `dfhack.job.printJobDetails(job)`

  Prints info about the job.

- `dfhack.job.printItemDetails(jobitem,idx)`

  Prints info about the job item.

- `dfhack.job.getGeneralRef(job, type)`

  Searches for a general_ref with the given type.

- `dfhack.job.getSpecificRef(job, type)`

  Searches for a specific_ref with the given type.

- `dfhack.job.getHolder(job)`

  Returns the building holding the job.

- `dfhack.job.getWorker(job)`

  Returns the unit performing the job.

- `dfhack.job.setJobCooldown(building,worker,timeout)`

  Prevent the worker from taking jobs at the specified workshop for the specified time. This doesn't decrease the timeout in any circumstances.

- `dfhack.job.removeWorker(job,timeout)`

  Removes the worker from the specified workshop job, and sets the cooldown. Returns *true* on success.

- `dfhack.job.checkBuildingsNow()`

  Instructs the game to check buildings for jobs next frame and assign workers.

- `dfhack.job.checkDesignationsNow()`

  Instructs the game to check designations for jobs next frame and assign workers.

- `dfhack.job.is_equal(job1,job2)`

  Compares important fields in the job and nested item structures.

- `dfhack.job.is_item_equal(job_item1,job_item2)`

  Compares important fields in the job item structures.

- `dfhack.job.linkIntoWorld(job,new_id)`

  Adds job into `df.global.job_list`, and if new_id is true, then also sets its id and increases `df.global.job_next_id`

- `dfhack.job.listNewlyCreated(first_id)`

  Returns the current value of `df.global.job_next_id`, and if there are any jobs with `first_id <= id < job_next_id`, a lua list containing them.

- `dfhack.job.isSuitableItem(job_item, item_type, item_subtype)`

  Does basic sanity checks to verify if the suggested item type matches the flags in the job item.

- `dfhack.job.isSuitableMaterial(job_item, mat_type, mat_index)`

  Likewise, if replacing material.

- `dfhack.job.getName(job)`

  Returns the job's description, as seen in the Units and Jobs screens.

Units module

- `dfhack.units.getPosition(unit)`

  Returns true *x,y,z* of the unit, or *nil* if invalid; may be not equal to unit.pos if caged.
- `dfhack.units.getGeneralRef(unit, type)`

  Searches for a general_ref with the given type.
- `dfhack.units.getSpecificRef(unit, type)`

  Searches for a specific_ref with the given type.
- `dfhack.units.getContainer(unit)`

  Returns the container (cage) item or *nil*.
- `dfhack.units.setNickname(unit,nick)`

  Sets the unit's nickname properly.
- `dfhack.units.getVisibleName(unit)`

  Returns the language_name object visible in game, accounting for false identities.
- `dfhack.units.getIdentity(unit)`

  Returns the false identity of the unit if it has one, or *nil*.
- `dfhack.units.getNemesis(unit)`

  Returns the nemesis record of the unit if it has one, or *nil*.
- `dfhack.units.isHidingCurse(unit)`

  Checks if the unit hides improved attributes from its curse.
- `dfhack.units.getPhysicalAttrValue(unit, attr_type)`
- `dfhack.units.getMentalAttrValue(unit, attr_type)`

  Computes the effective attribute value, including curse effect.
- `dfhack.units.isCrazed(unit)`
- `dfhack.units.isOpposedToLife(unit)`
- `dfhack.units.hasExtravision(unit)`
- `dfhack.units.isBloodsucker(unit)`

  Simple checks of caste attributes that can be modified by curses.
- `dfhack.units.getMiscTrait(unit, type[, create])`

  Finds (or creates if requested) a misc trait object with the given id.
- `dfhack.units.isDead(unit)`

  The unit is completely dead and passive, or a ghost.
- `dfhack.units.isAlive(unit)`

  The unit isn't dead or undead.
- `dfhack.units.isSane(unit)`

  The unit is capable of rational action, i.e. not dead, insane, zombie, or active werewolf.

- `dfhack.units.isDwarf(unit)`

  The unit is of the correct race of the fortress.

- `dfhack.units.isCitizen(unit)`

  The unit is an alive sane citizen of the fortress; wraps the same checks the game uses to decide game-over by extinction.

- `dfhack.units.getAge(unit[,true_age])`

  Returns the age of the unit in years as a floating-point value. If `true_age` is true, ignores false identities.

- `dfhack.units.getNominalSkill(unit, skill[, use_rust])`

  Retrieves the nominal skill level for the given unit. If `use_rust` is *true*, subtracts the rust penalty.

- `dfhack.units.getEffectiveSkill(unit, skill)`

  Computes the effective rating for the given skill, taking into account exhaustion, pain etc.

- `dfhack.units.getExperience(unit, skill[, total])`

  Returns the experience value for the given skill. If `total` is true, adds experience implied by the current rating.

- `dfhack.units.computeMovementSpeed(unit)`

  Computes number of frames * 100 it takes the unit to move in its current state of mind and body.

- `dfhack.units.computeSlowdownFactor(unit)`

  Meandering and floundering in liquid introduces additional slowdown. It is random, but the function computes and returns the expected mean factor as a float.

- `dfhack.units.getNoblePositions(unit)`

  Returns a list of tables describing noble position assignments, or *nil*. Every table has fields `entity`, `assignment` and `position`.

- `dfhack.units.getProfessionName(unit[,ignore_noble,plural])`

  Retrieves the profession name using custom profession, noble assignments or raws. The `ignore_noble` boolean disables the use of noble positions.

- `dfhack.units.getCasteProfessionName(race,caste,prof_id[,plural])`

  Retrieves the profession name for the given race/caste using raws.

- `dfhack.units.getProfessionColor(unit[,ignore_noble])`

  Retrieves the color associated with the profession, using noble assignments or raws. The `ignore_noble` boolean disables the use of noble positions.

- `dfhack.units.getCasteProfessionColor(race,caste,prof_id)`

  Retrieves the profession color for the given race/caste using raws.

Items module

- `dfhack.items.getPosition(item)`

  Returns true *x,y,z* of the item, or *nil* if invalid; may be not equal to item.pos if in inventory.

- `dfhack.items.getDescription(item, type[, decorate])`

  Returns the string description of the item, as produced by the `getItemDescription` method. If decorate is true, also adds markings for quality and improvements.

- `dfhack.items.getGeneralRef(item, type)`

  Searches for a general_ref with the given type.

- `dfhack.items.getSpecificRef(item, type)`

  Searches for a specific_ref with the given type.

- `dfhack.items.getOwner(item)`

  Returns the owner unit or *nil*.

- `dfhack.items.setOwner(item,unit)`

  Replaces the owner of the item. If unit is *nil*, removes ownership. Returns *false* in case of error.

- `dfhack.items.getContainer(item)`

  Returns the container item or *nil*.

- `dfhack.items.getContainedItems(item)`

  Returns a list of items contained in this one.

- `dfhack.items.getHolderBuilding(item)`

  Returns the holder building or *nil*.

- `dfhack.items.getHolderUnit(item)`

  Returns the holder unit or *nil*.

- `dfhack.items.moveToGround(item,pos)`

  Move the item to the ground at position. Returns *false* if impossible.

- `dfhack.items.moveToContainer(item,container)`

  Move the item to the container. Returns *false* if impossible.

- `dfhack.items.moveToBuilding(item,building[,use_mode[,force_in_building])`

  Move the item to the building. Returns *false* if impossible.

  `use_mode` defaults to 0. If set to 2, the item will be treated as part of the building.

  If `force_in_building` is true, the item will be considered to be stored by the building (used for items temporarily used in traps in vanilla DF)

- `dfhack.items.moveToInventory(item,unit,use_mode,body_part)`

  Move the item to the unit inventory. Returns *false* if impossible.

- `dfhack.items.remove(item[, no_uncat])`

  Removes the item, and marks it for garbage collection unless `no_uncat` is true.

- `dfhack.items.makeProjectile(item)`

  Turns the item into a projectile, and returns the new object, or *nil* if impossible.

- `dfhack.items.isCasteMaterial(item_type)`

  Returns *true* if this item type uses a creature/caste pair as its material.

- `dfhack.items.getSubtypeCount(item_type)`

  Returns the number of raw-defined subtypes of the given item type, or *-1* if not applicable.

- `dfhack.items.getSubtypeDef(item_type, subtype)`

  Returns the raw definition for the given item type and subtype, or *nil* if invalid.

- `dfhack.items.getItemBaseValue(item_type, subtype, material, mat_index)`

  Calculates the base value for an item of the specified type and material.

- `dfhack.items.getValue(item)`

  Calculates the Basic Value of an item, as seen in the View Item screen.

Maps module

- `dfhack.maps.getSize()`

  Returns map size in blocks: *x, y, z*

- `dfhack.maps.getTileSize()`

  Returns map size in tiles: *x, y, z*

- `dfhack.maps.getBlock(x,y,z)`

  Returns a map block object for given x,y,z in local block coordinates.

- `dfhack.maps.isValidTilePos(coords)`, or `isValidTilePos(x,y,z)`

  Checks if the given df::coord or x,y,z in local tile coordinates are valid.

- `dfhack.maps.getTileBlock(coords)`, or `getTileBlock(x,y,z)`

  Returns a map block object for given df::coord or x,y,z in local tile coordinates.

- `dfhack.maps.ensureTileBlock(coords)`, or `ensureTileBlock(x,y,z)`

  Like `getTileBlock`, but if the block is not allocated, try creating it.

- `dfhack.maps.getTileType(coords)`, or `getTileType(x,y,z)`

  Returns the tile type at the given coordinates, or *nil* if invalid.

- `dfhack.maps.getTileFlags(coords)`, or `getTileFlags(x,y,z)`

  Returns designation and occupancy references for the given coordinates, or *nil, nil* if invalid.

- `dfhack.maps.getRegionBiome(region_coord2d)`, or `getRegionBiome(x,y)`

  Returns the biome info struct for the given global map region.

- `dfhack.maps.enableBlockUpdates(block[,flow,temperature])`

  Enables updates for liquid flow or temperature, unless already active.

- `dfhack.maps.spawnFlow(pos,type,mat_type,mat_index,dimension)`

  Spawns a new flow (i.e. steam/mist/dust/etc) at the given pos, and with the given parameters. Returns it, or *nil* if unsuccessful.

- `dfhack.maps.getGlobalInitFeature(index)`

  Returns the global feature object with the given index.

- `dfhack.maps.getLocalInitFeature(region_coord2d,index)`

  Returns the local feature object with the given region coords and index.

- `dfhack.maps.getTileBiomeRgn(coords)`, or `getTileBiomeRgn(x,y,z)`

  Returns *x, y* for use with `getRegionBiome`.

- `dfhack.maps.canWalkBetween(pos1, pos2)`

  Checks if a dwarf may be able to walk between the two tiles, using a pathfinding cache maintained by the game.

  Note: This cache is only updated when the game is unpaused, and thus can get out of date if doors are forbidden or unforbidden, or tools like *liquids* or *tiletypes* are used. It also cannot possibly take into account anything that depends on the actual units, like burrows, or the presence of invaders.

- `dfhack.maps.hasTileAssignment(tilemask)`

  Checks if the tile_bitmask object is not *nil* and contains any set bits; returns *true* or *false*.

- `dfhack.maps.getTileAssignment(tilemask,x,y)`

  Checks if the tile_bitmask object is not *nil* and has the relevant bit set; returns *true* or *false*.

- `dfhack.maps.setTileAssignment(tilemask,x,y,enable)`

  Sets the relevant bit in the tile_bitmask object to the *enable* argument.

- `dfhack.maps.resetTileAssignment(tilemask[,enable])`

  Sets all bits in the mask to the *enable* argument.

Burrows module

- `dfhack.burrows.findByName(name)`

  Returns the burrow pointer or *nil*.

- `dfhack.burrows.clearUnits(burrow)`

  Removes all units from the burrow.

- `dfhack.burrows.isAssignedUnit(burrow,unit)`

  Checks if the unit is in the burrow.

- `dfhack.burrows.setAssignedUnit(burrow,unit,enable)`

  Adds or removes the unit from the burrow.

- `dfhack.burrows.clearTiles(burrow)`

  Removes all tiles from the burrow.

- `dfhack.burrows.listBlocks(burrow)`

  Returns a table of map block pointers.

- `dfhack.burrows.isAssignedTile(burrow,tile_coord)`

  Checks if the tile is in burrow.

- `dfhack.burrows.setAssignedTile(burrow,tile_coord,enable)`

  Adds or removes the tile from the burrow. Returns *false* if invalid coords.

- `dfhack.burrows.isAssignedBlockTile(burrow,block,x,y)`

  Checks if the tile within the block is in burrow.

- `dfhack.burrows.setAssignedBlockTile(burrow,block,x,y,enable)`

  Adds or removes the tile from the burrow. Returns *false* if invalid coords.

Buildings module

General

- `dfhack.buildings.getGeneralRef(building, type)`

  Searches for a general_ref with the given type.

- `dfhack.buildings.getSpecificRef(building, type)`

  Searches for a specific_ref with the given type.

- `dfhack.buildings.setOwner(item,unit)`

  Replaces the owner of the building. If unit is *nil*, removes ownership. Returns *false* in case of error.

- `dfhack.buildings.getSize(building)`

  Returns *width, height, centerx, centery*.

- `dfhack.buildings.findAtTile(pos)`, or `findAtTile(x,y,z)`

  Scans the buildings for the one located at the given tile. Does not work on civzones. Warning: linear scan if the map tile indicates there are buildings at it.

- `dfhack.buildings.findCivzonesAt(pos)`, or `findCivzonesAt(x,y,z)`

  Scans civzones, and returns a lua sequence of those that touch the given tile, or *nil* if none.

- `dfhack.buildings.getCorrectSize(width, height, type, subtype, custom, direction)`

  Computes correct dimensions for the specified building type and orientation, using width and height for flexible dimensions. Returns *is_flexible, width, height, center_x, center_y*.

- `dfhack.buildings.checkFreeTiles(pos,size[,extents,change_extents,allow_occupied])`

  Checks if the rectangle defined by `pos` and `size`, and possibly extents, can be used for placing a building. If `change_extents` is true, bad tiles are removed from extents. If `allow_occupied`, the occupancy test is skipped.

- `dfhack.buildings.countExtentTiles(extents,defval)`

  Returns the number of tiles included by extents, or defval.

- `dfhack.buildings.containsTile(building, x, y[, room])`

  Checks if the building contains the specified tile, either directly, or as room.

- `dfhack.buildings.hasSupport(pos,size)`

  Checks if a bridge constructed at specified position would have support from terrain, and thus won't collapse if retracted.

- `dfhack.buildings.getStockpileContents(stockpile)`

  Returns a list of items stored on the given stockpile. Ignores empty bins, barrels, and wheelbarrows assigned as storage and transport for that stockpile.

Low-level Low-level building creation functions:

- `dfhack.buildings.allocInstance(pos, type, subtype, custom)`

  Creates a new building instance of given type, subtype and custom type, at specified position. Returns the object, or *nil* in case of an error.

- `dfhack.buildings.setSize(building, width, height, direction)`

  Configures an object returned by `allocInstance`, using specified parameters wherever appropriate. If the building has fixed size along any dimension, the corresponding input parameter will be ignored. Returns *false* if the building cannot be placed, or *true, width, height, rect_area, true_area*. Returned width and height are the final values used by the building; true_area is less than rect_area if any tiles were removed from designation.

- `dfhack.buildings.constructAbstract(building)`

  Links a fully configured object created by `allocInstance` into the world. The object must be an abstract building, i.e. a stockpile or civzone. Returns *true*, or *false* if impossible.

- `dfhack.buildings.constructWithItems(building, items)`

  Links a fully configured object created by `allocInstance` into the world for construction, using a list of specific items as material. Returns *true*, or *false* if impossible.

- `dfhack.buildings.constructWithFilters(building, job_items)`

  Links a fully configured object created by `allocInstance` into the world for construction, using a list of job_item filters as inputs. Returns *true*, or *false* if impossible. Filter objects are claimed and possibly destroyed in any case. Use a negative `quantity` field value to auto-compute the amount from the size of the building.

- `dfhack.buildings.deconstruct(building)`

  Destroys the building, or queues a deconstruction job. Returns *true* if the building was destroyed and deallocated immediately.

High-level More high-level functions are implemented in lua and can be loaded by `require('dfhack.buildings')`. See `hack/lua/dfhack/buildings.lua`.

Among them are:

- `dfhack.buildings.getFiltersByType(argtable,type,subtype,custom)`

  Returns a sequence of lua structures, describing input item filters suitable for the specified building type, or *nil* if unknown or invalid. The returned sequence is suitable for use as the `job_items` argument of `constructWithFilters`. Uses tables defined in `buildings.lua`.

  Argtable members `material` (the default name), `bucket`, `barrel`, `chain`, `mechanism`, `screw`, `pipe`, `anvil`, `weapon` are used to augment the basic attributes with more detailed information if the building has input items with the matching name (see the tables for naming details). Note that it is impossible to *override* any properties this way, only supply those that are not mentioned otherwise; one exception is that flags2.non_economic is automatically cleared if an explicit material is specified.

- `dfhack.buildings.constructBuilding{...}`

  Creates a building in one call, using options contained in the argument table. Returns the building, or *nil, error*.

  Note: Despite the name, unless the building is abstract, the function creates it in an 'unconstructed' stage, with a queued in-game job that will actually construct it. I.e. the function replicates programmatically what can be done through the construct building menu in the game ui, except that it does less environment constraint checking.

The following options can be used:

- `pos = coordinates`, or `x = ..., y = ..., z = ...`

  Mandatory. Specifies the left upper corner of the building.

- `type = df.building_type.FOO, subtype = ..., custom = ...`

  Mandatory. Specifies the type of the building. Obviously, subtype and custom are only expected if the type requires them.

- `fields = { ... }`

  Initializes fields of the building object after creation with `df.assign`.

- `width = ..., height = ..., direction = ...`

  Sets size and orientation of the building. If it is fixed-size, specified dimensions are ignored.

- `full_rectangle = true`

  For buildings like stockpiles or farm plots that can normally accomodate individual tile exclusion, forces an error if any tiles within the specified width*height are obstructed.

- `items = { item, item ... }`, or `filters = { {...}, {...}... }`

  Specifies explicit items or item filters to use in construction. It is the job of the user to ensure they are correct for the building type.

- `abstract = true`

  Specifies that the building is abstract and does not require construction. Required for stockpiles and civzones; an error otherwise.

- `material = {...}, mechanism = {...}, ...`

  If none of `items`, `filter`, or `abstract` is used, the function uses `getFiltersByType` to compute the input item filters, and passes the argument table through. If no filters can be determined this way, `constructBuilding` throws an error.

Constructions module

- `dfhack.constructions.designateNew(pos,type,item_type,mat_index)`

  Designates a new construction at given position. If there already is a planned but not completed construction there, changes its type. Returns *true*, or *false* if obstructed. Note that designated constructions are technically buildings.

- `dfhack.constructions.designateRemove(pos)`, or `designateRemove(x,y,z)`

  If there is a construction or a planned construction at the specified coordinates, designates it for removal, or instantly cancels the planned one. Returns *true, was_only_planned* if removed; or *false* if none found.

Screen API

The screen module implements support for drawing to the tiled screen of the game. Note that drawing only has any effect when done from callbacks, so it can only be feasibly used in the core context.

Basic painting functions:

- `dfhack.screen.getWindowSize()`

  Returns *width, height* of the screen.

- `dfhack.screen.getMousePos()`

  Returns *x,y* of the tile the mouse is over.

- `dfhack.screen.inGraphicsMode()`

  Checks if [GRAPHICS:YES] was specified in init.

- `dfhack.screen.paintTile(pen,x,y[,char,tile])`

  Paints a tile using given parameters. See below for a description of pen.

  Returns *false* if coordinates out of bounds, or other error.

- `dfhack.screen.readTile(x,y)`

  Retrieves the contents of the specified tile from the screen buffers. Returns a pen object, or *nil* if invalid or TrueType.

- `dfhack.screen.paintString(pen,x,y,text)`

  Paints the string starting at *x,y*. Uses the string characters in sequence to override the `ch` field of pen.

  Returns *true* if painting at least one character succeeded.

- `dfhack.screen.fillRect(pen,x1,y1,x2,y2)`

  Fills the rectangle specified by the coordinates with the given pen. Returns *true* if painting at least one character succeeded.

- `dfhack.screen.findGraphicsTile(pagename,x,y)`

  Finds a tile from a graphics set (i.e. the raws used for creatures), if in graphics mode and loaded.

  Returns: *tile, tile_grayscale*, or *nil* if not found. The values can then be used for the *tile* field of *pen* structures.

- `dfhack.screen.clear()`

  Fills the screen with blank background.

- `dfhack.screen.invalidate()`

  Requests repaint of the screen by setting a flag. Unlike other functions in this section, this may be used at any time.

- `dfhack.screen.getKeyDisplay(key)`

  Returns the string that should be used to represent the given logical keybinding on the screen in texts like "press Key to ...".

- `dfhack.screen.keyToChar(key)`

  Returns the integer character code of the string input character represented by the given logical keybinding, or *nil* if not a string input key.

- `dfhack.screen.charToKey(charcode)`

  Returns the keybinding representing the given string input character, or *nil* if impossible.

The "pen" argument used by functions above may be represented by a table with the following possible fields:

`ch` Provides the ordinary tile character, as either a 1-character string or a number. Can be overridden with the `char` function parameter.

`fg` Foreground color for the ordinary tile. Defaults to COLOR_GREY (7).

`bg` Background color for the ordinary tile. Defaults to COLOR_BLACK (0).

bold Bright/bold text flag. If *nil*, computed based on (fg & 8); fg is masked to 3 bits. Otherwise should be *true/false*.

tile Graphical tile id. Ignored unless [GRAPHICS:YES] was in init.txt.

tile_color = true Specifies that the tile should be shaded with *fg/bg*.

tile_fg, tile_bg If specified, overrides *tile_color* and supplies shading colors directly.

Alternatively, it may be a pre-parsed native object with the following API:

- `dfhack.pen.make(base[,pen_or_fg,bg,bold])`

  Creates a new pre-parsed pen by combining its arguments according to the following rules:

  1. The `base` argument may be a pen object, a pen table as specified above, or a single color value. In the single value case, it is split into `fg` and `bold` properties, and others are initialized to 0. This argument will be converted to a pre-parsed object and returned if there are no other arguments.
  2. If the `pen_or_fg` argument is specified as a table or object, it completely replaces the base, and is returned instead of it.
  3. Otherwise, the non-nil subset of the optional arguments is used to update the `fg`, `bg` and `bold` properties of the base. If the `bold` flag is *nil*, but *pen_or_fg* is a number, `bold` is deduced from it like in the simple base case.

  This function always returns a new pre-parsed pen, or *nil*.

- `dfhack.pen.parse(base[,pen_or_fg,bg,bold])`

  Exactly like the above function, but returns `base` or `pen_or_fg` directly if they are already a pre-parsed native object.

- `pen.property`, `pen.property = value`, `pairs(pen)`

  Pre-parsed pens support reading and setting their properties, but don't behave exactly like a simple table would; for instance, assigning to `pen.tile_color` also resets `pen.tile_fg` and `pen.tile_bg` to *nil*.

In order to actually be able to paint to the screen, it is necessary to create and register a viewscreen (basically a modal dialog) with the game.

Warning: As a matter of policy, in order to avoid user confusion, all interface screens added by dfhack should bear the "DFHack" signature.

Screens are managed with the following functions:

- `dfhack.screen.show(screen[,below])`

  Displays the given screen, possibly placing it below a different one. The screen must not be already shown. Returns *true* if success.

- `dfhack.screen.dismiss(screen[,to_first])`

  Marks the screen to be removed when the game enters its event loop. If `to_first` is *true*, all screens up to the first one will be deleted.

- `dfhack.screen.isDismissed(screen)`

  Checks if the screen is already marked for removal.

Apart from a native viewscreen object, these functions accept a table as a screen. In this case, `show` creates a new native viewscreen that delegates all processing to methods stored in that table.

Note: Lua-implemented screens are only supported in the core context.

Supported callbacks and fields are:

- `screen._native`

  Initialized by `show` with a reference to the backing viewscreen object, and removed again when the object is deleted.

- `function screen:onShow()`

  Called by `dfhack.screen.show` if successful.

- `function screen:onDismiss()`

  Called by `dfhack.screen.dismiss` if successful.

- `function screen:onDestroy()`

  Called from the destructor when the viewscreen is deleted.

- `function screen:onResize(w, h)`

  Called before `onRender` or `onIdle` when the window size has changed.

- `function screen:onRender()`

  Called when the viewscreen should paint itself. This is the only context where the above painting functions work correctly.

  If omitted, the screen is cleared; otherwise it should do that itself. In order to make a see-through dialog, call `self._native.parent:render()`.

- `function screen:onIdle()`

  Called every frame when the screen is on top of the stack.

- `function screen:onHelp()`

  Called when the help keybinding is activated (usually '?').

- `function screen:onInput(keys)`

  Called when keyboard or mouse events are available. If any keys are pressed, the keys argument is a table mapping them to *true*. Note that this refers to logical keybingings computed from real keys via options; if multiple interpretations exist, the table will contain multiple keys.

  The table also may contain special keys:

  `_STRING` Maps to an integer in range 0-255. Duplicates a separate "STRING_A???" code for convenience.

  `_MOUSE_L, _MOUSE_R` If the left or right mouse button is being pressed.

  `_MOUSE_L_DOWN, _MOUSE_R_DOWN` If the left or right mouse button was just pressed.

  If this method is omitted, the screen is dismissed on receival of the `LEAVESCREEN` key.

- `function screen:onGetSelectedUnit()`
- `function screen:onGetSelectedItem()`
- `function screen:onGetSelectedJob()`
- `function screen:onGetSelectedBuilding()`

  Implement these to provide a return value for the matching `dfhack.gui.getSelected...` function.

PenArray class

Screens that require significant computation in their onRender() method can use a `dfhack.penarray` instance to cache their output.

- `dfhack.penarray.new(w, h)`

  Creates a new penarray instance with an internal buffer of `w * h` tiles. These dimensions currently cannot be changed after a penarray is instantiated.

- `penarray:clear()`

  Clears the internal buffer, similar to `dfhack.screen.clear()`.

- `penarray:get_dims()`

  Returns the x and y dimensions of the internal buffer.

- `penarray:get_tile(x, y)`

  Returns a pen corresponding to the tile at (`x`, `y`) in the internal buffer. Note that indices are 0-based.

- `penarray:set_tile(x, y, pen)`

  Sets the tile at (`x`, `y`) in the internal buffer to the pen given.

- `penarray:draw(x, y, w, h, bufferx, buffery)`

  Draws the contents of the internal buffer, beginning at (`bufferx`, `buffery`) and spanning `w` columns and `h` rows, to the screen starting at (`x`, `y`). Any invalid screen and buffer coordinates are skipped.

  `bufferx` and `buffery` default to 0.

Filesystem module

Most of these functions return `true` on success and `false` on failure, unless otherwise noted.

- `dfhack.filesystem.exists(path)`

  Returns `true` if `path` exists.

- `dfhack.filesystem.isfile(path)`

  Returns `true` if `path` exists and is a file.

- `dfhack.filesystem.isdir(path)`

  Returns `true` if `path` exists and is a directory.

- `dfhack.filesystem.getcwd()`

  Returns the current working directory. To retrieve the DF path, use `dfhack.getDFPath()` instead.

- `dfhack.filesystem.chdir(path)`

  Changes the current directory to `path`. Use with caution.

- `dfhack.filesystem.mkdir(path)`

  Creates a new directory. Returns `false` if unsuccessful, including if `path` already exists.

- `dfhack.filesystem.rmdir(path)`

  Removes a directory. Only works if the directory is already empty.

- `dfhack.filesystem.mtime(path)`

  Returns the modification time (in seconds) of the file or directory specified by `path`, or -1 if `path` does not exist. This depends on the system clock and should only be used locally.

- `dfhack.filesystem.atime(path)`
- `dfhack.filesystem.ctime(path)`

  Return values vary across operating systems - return the `st_atime` and `st_ctime` fields of a C++ stat struct, respectively.

- `dfhack.filesystem.listdir(path)`

  Lists files/directories in a directory. Returns `{}` if `path` does not exist.

- `dfhack.filesystem.listdir_recursive(path [, depth = 10])`

  Lists all files/directories in a directory and its subdirectories. All directories are listed before their contents. Returns a table with subtables of the format:

```
{path: 'path to file', isdir: true|false}
```

  Note that `listdir()` returns only the base name of each directory entry, while `listdir_recursive()` returns the initial path and all components following it for each entry.

Internal API

These functions are intended for the use by dfhack developers, and are only documented here for completeness:

- `dfhack.internal.scripts`

  The table used by `dfhack.run_script()` to give every script its own global environment, persistent between calls to the script.

- `dfhack.internal.getPE()`

  Returns the PE timestamp of the DF executable (only on Windows)

- `dfhack.internal.getMD5()`

  Returns the MD5 of the DF executable (only on OS X and Linux)

- `dfhack.internal.getAddress(name)`

  Returns the global address `name`, or *nil*.

- `dfhack.internal.setAddress(name, value)`

  Sets the global address `name`. Returns the value of `getAddress` before the change.

- `dfhack.internal.getVTable(name)`

  Returns the pre-extracted vtable address `name`, or *nil*.

- `dfhack.internal.getImageBase()`

  Returns the mmap base of the executable.

- `dfhack.internal.getRebaseDelta()`

  Returns the ASLR rebase offset of the DF executable.

- `dfhack.internal.adjustOffset(offset[,to_file])`

  Returns the re-aligned offset, or *nil* if invalid. If `to_file` is true, the offset is adjusted from memory to file. This function returns the original value everywhere except windows.

- `dfhack.internal.getMemRanges()`

  Returns a sequence of tables describing virtual memory ranges of the process.

- `dfhack.internal.patchMemory(dest,src,count)`

  Like memmove below, but works even if dest is read-only memory, e.g. code. If destination overlaps a completely invalid memory region, or another error occurs, returns false.

- `dfhack.internal.patchBytes(write_table[, verify_table])`

  The first argument must be a lua table, which is interpreted as a mapping from memory addresses to byte values that should be stored there. The second argument may be a similar table of values that need to be checked before writing anything.

  The function takes care to either apply all of `write_table`, or none of it. An empty `write_table` with a nonempty `verify_table` can be used to reasonably safely check if the memory contains certain values.

  Returns *true* if successful, or *nil, error_msg, address* if not.

- `dfhack.internal.memmove(dest,src,count)`

  Wraps the standard memmove function. Accepts both numbers and refs as pointers.

- `dfhack.internal.memcmp(ptr1,ptr2,count)`

  Wraps the standard memcmp function.

- `dfhack.internal.memscan(haystack,count,step,needle,nsize)`

  Searches for `needle` of `nsize` bytes in `haystack`, using `count` steps of `step` bytes. Returns: *step_idx, sum_idx, found_ptr*, or *nil* if not found.

- `dfhack.internal.diffscan(old_data, new_data, start_idx, end_idx, eltsize[, oldval, newval, delta])`

  Searches for differences between buffers at ptr1 and ptr2, as integers of size eltsize. The oldval, newval or delta arguments may be used to specify additional constraints. Returns: *found_index*, or *nil* if end reached.

- `dfhack.internal.getDir(path)`

  Lists files/directories in a directory. Returns: *file_names* or empty table if not found. Identical to `dfhack.filesystem.listdir(path)`.

- `dfhack.internal.strerror(errno)`

  Wraps strerror() - returns a string describing a platform-specific error code

- `dfhack.internal.addScriptPath(path, search_before)`

  Adds `path` to the list of paths searched for scripts (both in Lua and Ruby). If `search_before` is passed and `true`, the path will be searched before the default paths (e.g. `raw/scripts`, `hack/scripts`); otherwise, it will be searched after.

  Returns `true` if successful or `false` otherwise (e.g. if the path does not exist or has already been registered).

- `dfhack.internal.removeScriptPath(path)`

  Removes `path` from the script search paths and returns `true` if successful.

- `dfhack.internal.getScriptPaths()`

  Returns the list of script paths in the order they are searched, including defaults. (This can change if a world is loaded.)

- `dfhack.internal.findScript(name)`

  Searches script paths for the script `name` and returns the path of the first file found, or `nil` on failure.

  Note: This requires an extension to be specified (`.lua` or `.rb`) - use `dfhack.findScript()` to include the `.lua` extension automatically.

Core interpreter context

While plugins can create any number of interpreter instances, there is one special context managed by dfhack core. It is the only context that can receive events from DF and plugins.

Core context specific functions:

- `dfhack.is_core_context`

  Boolean value; *true* in the core context.

- `dfhack.timeout(time,mode,callback)`

  Arranges for the callback to be called once the specified period of time passes. The `mode` argument specifies the unit of time used, and may be one of `'frames'` (raw FPS), `'ticks'` (unpaused FPS), `'days'`, `'months'`, `'years'` (in-game time). All timers other than `'frames'` are cancelled when the world is unloaded, and cannot be queued until it is loaded again. Returns the timer id, or *nil* if unsuccessful due to world being unloaded.

- `dfhack.timeout_active(id[,new_callback])`

  Returns the active callback with the given id, or *nil* if inactive or nil id. If called with 2 arguments, replaces the current callback with the given value, if still active. Using `timeout_active(id,nil)` cancels the timer.

- `dfhack.onStateChange.foo = function(code)`

  Event. Receives the same codes as plugin_onstatechange in C++.

Event type

An event is a native object transparently wrapping a lua table, and implementing a __call metamethod. When it is invoked, it loops through the table with next and calls all contained values. This is intended as an extensible way to add listeners.

This type itself is available in any context, but only the core context has the actual events defined by C++ code.

Features:

- `dfhack.event.new()`

  Creates a new instance of an event.

- `event[key] = function`

  Sets the function as one of the listeners. Assign *nil* to remove it.

  Note: The `df.NULL` key is reserved for the use by the C++ owner of the event; it is an error to try setting it.

- `#event`

  Returns the number of non-nil listeners.

- `pairs(event)`

  Iterates over all listeners in the table.

- `event(args...)`

  Invokes all listeners contained in the event in an arbitrary order using `dfhack.safecall`.

4.3.3 Lua Modules

DFHack sets up the lua interpreter so that the built-in `require` function can be used to load shared lua code from `hack/lua/`. The `dfhack` namespace reference itself may be obtained via `require('dfhack')`, although it is initially created as a global by C++ bootstrap code.

The following module management functions are provided:

- `mkmodule(name)`

  Creates an environment table for the module. Intended to be used as:

```
local _ENV = mkmodule('foo')
...
return _ENV
```

  If called the second time, returns the same table; thus providing reload support.

- `reload(name)`

  Reloads a previously `require`-d module *"name"* from the file. Intended as a help for module development.

- `dfhack.BASE_G`

  This variable contains the root global environment table, which is used as a base for all module and script environments. Its contents should be kept limited to the standard Lua library and API described in this document.

Global environment

A number of variables and functions are provided in the base global environment by the mandatory init file dfhack.lua:

- Color constants

  These are applicable both for `dfhack.color()` and color fields in DF functions or structures:

```
COLOR_RESET, COLOR_BLACK, COLOR_BLUE, COLOR_GREEN, COLOR_CYAN,
COLOR_RED, COLOR_MAGENTA, COLOR_BROWN, COLOR_GREY, COLOR_DARKGREY,
COLOR_LIGHTBLUE, COLOR_LIGHTGREEN, COLOR_LIGHTCYAN, COLOR_LIGHTRED,
COLOR_LIGHTMAGENTA, COLOR_YELLOW, COLOR_WHITE
```

- `dfhack.onStateChange` event codes

  Available only in the core context, as is the event itself:

  SC_WORLD_LOADED, SC_WORLD_UNLOADED, SC_MAP_LOADED, SC_MAP_UNLOADED, SC_VIEWSCREEN_CHANGED, SC_CORE_INITIALIZED

- Functions already described above

  safecall, qerror, mkmodule, reload

- Miscellaneous constants

  NEWLINE, COMMA, PERIOD evaluate to the relevant character strings.

  DEFAULT_NIL is an unspecified unique token used by the class module below.

- `printall(obj)`

  If the argument is a lua table or DF object reference, prints all fields.

- `copyall(obj)`

  Returns a shallow copy of the table or reference as a lua table.

- `pos2xyz(obj)`

  The object must have fields x, y and z. Returns them as 3 values. If obj is *nil*, or x is -30000 (the usual marker for undefined coordinates), returns *nil*.

- `xyz2pos(x,y,z)`

  Returns a table with x, y and z as fields.

- `same_xyz(a,b)`

  Checks if `a` and `b` have the same x, y and z fields.

- `get_path_xyz(path,i)`

  Returns `path.x[i], path.y[i], path.z[i]`.

- `pos2xy(obj)`, `xy2pos(x,y)`, `same_xy(a,b)`, `get_path_xy(a,b)`

  Same as above, but for 2D coordinates.

- `safe_index(obj,index...)`

  Walks a sequence of dereferences, which may be represented by numbers or strings. Returns *nil* if any of obj or indices is *nil*, or a numeric index is out of array bounds.

utils

- `utils.compare(a,b)`

  Comparator function; returns *-1* if a<b, *1* if a>b, *0* otherwise.

- `utils.compare_name(a,b)`

  Comparator for names; compares empty string last.

- `utils.is_container(obj)`

  Checks if obj is a container ref.

- `utils.make_index_sequence(start,end)`

  Returns a lua sequence of numbers in start..end.

- `utils.make_sort_order(data, ordering)`

  Computes a sorted permutation of objects in data, as a table of integer indices into the data sequence. Uses `data.n` as input length if present.

  The ordering argument is a sequence of ordering specs, represented as lua tables with following possible fields:

ord.key = *function(value)* Computes comparison key from input data value. Not called on nil. If omitted, the comparison key is the value itself.

ord.key_table = *function(data)* Computes a key table from the data table in one go.

ord.compare = *function(a,b)* Comparison function. Defaults to `utils.compare` above. Called on non-nil keys; nil sorts last.

ord.nil_first = *true/false* If true, nil keys are sorted first instead of last.

ord.reverse = *true/false* If true, sort non-nil keys in descending order.

For every comparison during sorting the specs are applied in order until an unambiguous decision is reached. Sorting is stable.

Example of sorting a sequence by field foo:

```
local spec = { key = function(v) return v.foo end }
local order = utils.make_sort_order(data, { spec })
local output = {}
for i = 1,#order do output[i] = data[order[i]] end
```

Separating the actual reordering of the sequence in this way enables applying the same permutation to multiple arrays. This function is used by the sort plugin.

- `for link,item in utils.listpairs(list)`

  Iterates a df-list structure, for example `df.global.world.job_list`.

- `utils.assign(tgt, src)`

  Does a recursive assignment of src into tgt. Uses `df.assign` if tgt is a native object ref; otherwise recurses into lua tables.

- `utils.clone(obj, deep)`

  Performs a shallow, or semi-deep copy of the object as a lua table tree. The deep mode recurses into lua tables and subobjects, except pointers to other heap objects. Null pointers are represented as `df.NULL`. Zero-based native containers are converted to 1-based lua sequences.

- `utils.clone_with_default(obj, default, force)`

  Copies the object, using the `default` lua table tree as a guide to which values should be skipped as uninteresting. The `force` argument makes it always return a non-*nil* value.

- `utils.parse_bitfield_int(value, type_ref)`

  Given an int `value`, and a bitfield type in the `df` tree, it returns a lua table mapping the enabled bit keys to *true*, unless value is 0, in which case it returns *nil*.

- `utils.list_bitfield_flags(bitfield[, list])`

  Adds all enabled bitfield keys to `list` or a newly-allocated empty sequence, and returns it. The `bitfield` argument may be *nil*.

- `utils.sort_vector(vector,field,cmpfun)`

  Sorts a native vector or lua sequence using the comparator function. If `field` is not *nil*, applies the comparator to the field instead of the whole object.

- `utils.linear_index(vector,key[,field])`

  Searches for `key` in the vector, and returns *index, found_value*, or *nil* if none found.

- `utils.binsearch(vector,key,field,cmpfun,min,max)`

  Does a binary search in a native vector or lua sequence for `key`, using `cmpfun` and `field` like sort_vector. If `min` and `max` are specified, they are used as the search subrange bounds.

  If found, returns *item, true, idx*. Otherwise returns *nil, false, insert_idx*, where *insert_idx* is the correct insertion point.

- `utils.insert_sorted(vector,item,field,cmpfun)`

  Does a binary search, and inserts item if not found. Returns *did_insert, vector[idx], idx*.

- `utils.insert_or_update(vector,item,field,cmpfun)`

  Like `insert_sorted`, but also assigns the item into the vector cell if insertion didn't happen.

  As an example, you can use this to set skill values:

```
utils.insert_or_update(soul.skills, {new=true, id=..., rating=...}, 'id')
```

  (For an explanation of `new=true`, see *Recursive table assignment*)

- `utils.erase_sorted_key(vector,key,field,cmpfun)`

  Removes the item with the given key from the list. Returns: *did_erase, vector[idx], idx*.

- `utils.erase_sorted(vector,item,field,cmpfun)`

  Exactly like `erase_sorted_key`, but if field is specified, takes the key from `item[field]`.

- `utils.call_with_string(obj,methodname,...)`

  Allocates a temporary string object, calls `obj:method(tmp,...)`, and returns the value written into the temporary after deleting it.

- `utils.getBuildingName(building)`

  Returns the string description of the given building.

- `utils.getBuildingCenter(building)`

  Returns an x/y/z table pointing at the building center.

- `utils.split_string(string, delimiter)`

  Splits the string by the given delimiter, and returns a sequence of results.

- `utils.prompt_yes_no(prompt, default)`

  Presents a yes/no prompt to the user. If `default` is not *nil*, allows just pressing Enter to submit the default choice. If the user enters `'abort'`, throws an error.

- `utils.prompt_input(prompt, checkfun, quit_str)`

  Presents a prompt to input data, until a valid string is entered. Once `checkfun(input)` returns *true, ...*, passes the values through. If the user enters the quit_str (defaults to `'~~~'`), throws an error.

- `utils.check_number(text)`

  A `prompt_input` `checkfun` that verifies a number input.

dumper

A third-party lua table dumper module from http://lua-users.org/wiki/DataDumper. Defines one function:

- `dumper.DataDumper(value, varname, fastmode, ident, indent_step)`

  Returns `value` converted to a string. The `indent_step` argument specifies the indentation step size in spaces. For the other arguments see the original documentation link above.

class

Implements a trivial single-inheritance class system.

- `Foo = defclass(Foo[, ParentClass])`

  Defines or updates class Foo. The `Foo = defclass(Foo)` syntax is needed so that when the module or script is reloaded, the class identity will be preserved through the preservation of global variable values.

  The `defclass` function is defined as a stub in the global namespace, and using it will auto-load the class module.

- `Class.super`

  This class field is set by defclass to the parent class, and allows a readable `Class.super.method(self, ...)` syntax for calling superclass methods.

- `Class.ATTRS { foo = xxx, bar = yyy }`

  Declares certain instance fields to be attributes, i.e. auto-initialized from fields in the table used as the constructor argument. If omitted, they are initialized with the default values specified in this declaration.

  If the default value should be *nil*, use `ATTRS { foo = DEFAULT_NIL }`.

  Declaring an attribute is mostly the same as defining your `init` method like this:

```
function Class.init(args)
    self.attr1 = args.attr1 or default1
    self.attr2 = args.attr2 or default2
    ...
end
```

  The main difference is that attributes are processed as a separate initialization step, before any `init` methods are called. They also make the directy relation between instance fields and constructor arguments more explicit.

- `new_obj = Class{ foo = arg, bar = arg, ... }`

  Calling the class as a function creates and initializes a new instance. Initialization happens in this order:

  1. An empty instance table is created, and its metatable set.
  2. The `preinit` methods are called via `invoke_before` (see below) with the table used as argument to the class. These methods are intended for validating and tweaking that argument table.
  3. Declared ATTRS are initialized from the argument table or their default values.
  4. The `init` methods are called via `invoke_after` with the argument table. This is the main constructor method.
  5. The `postinit` methods are called via `invoke_after` with the argument table. Place code that should be called after the object is fully constructed here.

Predefined instance methods:

- `instance:assign{ foo = xxx }`

  Assigns all values in the input table to the matching instance fields.

- `instance:callback(method_name, [args...])`

  Returns a closure that invokes the specified method of the class, properly passing in self, and optionally a number of initial arguments too. The arguments given to the closure are appended to these.

- `instance:cb_getfield(field_name)`

  Returns a closure that returns the specified field of the object when called.

- `instance:cb_setfield(field_name)`

  Returns a closure that sets the specified field to its argument when called.

- `instance:invoke_before(method_name, args...)`

  Navigates the inheritance chain of the instance starting from the most specific class, and invokes the specified method with the arguments if it is defined in that specific class. Equivalent to the following definition in every class:

```
function Class:invoke_before(method, ...)
  if rawget(Class, method) then
    rawget(Class, method)(self, ...)
  end
  Class.super.invoke_before(method, ...)
end
```

- `instance:invoke_after(method_name, args...)`

  Like invoke_before, only the method is called after the recursive call to super, i.e. invocations happen in the parent to child order.

  These two methods are inspired by the Common Lisp before and after methods, and are intended for implementing similar protocols for certain things. The class library itself uses them for constructors.

To avoid confusion, these methods cannot be redefined.

4.3.4 In-game UI Library

A number of lua modules with names starting with `gui` are dedicated to wrapping the natives of the `dfhack.screen` module in a way that is easy to use. This allows relatively easily and naturally creating dialogs that integrate in the main game UI window.

These modules make extensive use of the `class` module, and define things ranging from the basic `Painter`, `View` and `Screen` classes, to fully functional predefined dialogs.

gui

This module defines the most important classes and functions for implementing interfaces. This documents those of them that are considered stable.

Misc

- `USE_GRAPHICS`

  Contains the value of `dfhack.screen.inGraphicsMode()`, which cannot be changed without restarting the game and thus is constant during the session.

- `CLEAR_PEN`

  The black pen used to clear the screen.

- `simulateInput(screen, keys...)`

  This function wraps an undocumented native function that passes a set of keycodes to a screen, and is the official way to do that.

  Every argument after the initial screen may be *nil*, a numeric keycode, a string keycode, a sequence of numeric or string keycodes, or a mapping of keycodes to *true* or *false*. For instance, it is possible to use the table passed as argument to `onInput`.

- `mkdims_xy(x1,y1,x2,y2)`

  Returns a table containing the arguments as fields, and also `width` and `height` that contains the rectangle dimensions.

- `mkdims_wh(x1,y1,width,height)`

  Returns the same kind of table as `mkdims_xy`, only this time it computes `x2` and `y2`.

- `is_in_rect(rect,x,y)`

  Checks if the given point is within a rectangle, represented by a table produced by one of the `mkdims` functions.

- `blink_visible(delay)`

  Returns *true* or *false*, with the value switching to the opposite every `delay` msec. This is intended for rendering blinking interface objects.

- `getKeyDisplay(keycode)`

  Wraps `dfhack.screen.getKeyDisplay` in order to allow using strings for the keycode argument.

ViewRect class

This class represents an on-screen rectangle with an associated independent clip area rectangle. It is the base of the `Painter` class, and is used by `Views` to track their client area.

- `ViewRect{ rect = ..., clip_rect = ..., view_rect = ..., clip_view = ... }`

  The constructor has the following arguments:

  rect The `mkdims` rectangle in screen coordinates of the logical viewport. Defaults to the whole screen.

clip_rect The clip rectangle in screen coordinates. Defaults to `rect`.

view_rect A ViewRect object to copy from; overrides both `rect` and `clip_rect`.

clip_view A ViewRect object to intersect the specified clip area with.

- `rect:isDefunct()`

  Returns *true* if the clip area is empty, i.e. no painting is possible.

- `rect:inClipGlobalXY(x,y)`

  Checks if these global coordinates are within the clip rectangle.

- `rect:inClipLocalXY(x,y)`

  Checks if these coordinates (specified relative to `x1,y1`) are within the clip rectangle.

- `rect:localXY(x,y)`

  Converts a pair of global coordinates to local; returns *x_local,y_local*.

- `rect:globalXY(x,y)`

  Converts a pair of local coordinates to global; returns *x_global,y_global*.

- `rect:viewport(x,y,w,h)` or `rect:viewport(subrect)`

  Returns a ViewRect representing a sub-rectangle of the current one. The arguments are specified in local coordinates; the `subrect` argument must be a `mkdims` table. The returned object consists of the exact specified rectangle, and a clip area produced by intersecting it with the clip area of the original object.

Painter class

The painting natives in `dfhack.screen` apply to the whole screen, are completely stateless and don't implement clipping.

The Painter class inherits from ViewRect to provide clipping and local coordinates, and tracks current cursor position and current pen.

- `Painter{ ..., pen = ..., key_pen = ...  }`

  In addition to ViewRect arguments, Painter accepts a suggestion of the initial value for the main pen, and the keybinding pen. They default to COLOR_GREY and COLOR_LIGHTGREEN otherwise.

  There are also some convenience functions that wrap this constructor:

  - `Painter.new(rect,pen)`
  - `Painter.new_view(view_rect,pen)`
  - `Painter.new_xy(x1,y1,x2,y2,pen)`
  - `Painter.new_wh(x1,y1,width,height,pen)`

- `painter:isValidPos()`

  Checks if the current cursor position is within the clip area.

- `painter:viewport(x,y,w,h)`

  Like the superclass method, but returns a Painter object.

- `painter:cursor()`

  Returns the current cursor *x,y* in local coordinates.

- `painter:seek(x,y)`

  Sets the current cursor position, and returns *self*. Either of the arguments may be *nil* to keep the current value.

- `painter:advance(dx,dy)`

  Adds the given offsets to the cursor position, and returns *self*. Either of the arguments may be *nil* to keep the current value.

- `painter:newline([dx])`

  Advances the cursor to the start of the next line plus the given x offset, and returns *self*.

- `painter:pen(...)`

  Sets the current pen to `dfhack.pen.parse(old_pen,...)`, and returns *self*.

- `painter:key_pen(...)`

  Sets the current keybinding pen to `dfhack.pen.parse(old_pen,...)`, and returns *self*.

- `painter:clear()`

  Fills the whole clip rectangle with `CLEAR_PEN`, and returns *self*.

- `painter:fill(x1,y1,x2,y2[,...])` or `painter:fill(rect[,...])`

  Fills the specified local coordinate rectangle with `dfhack.pen.parse(cur_pen,...)`, and returns *self*.

- `painter:char([char[, ...]])`

  Paints one character using `char` and `dfhack.pen.parse(cur_pen,...)`; returns *self*. The `char` argument, if not nil, is used to override the `ch` property of the pen.

- `painter:tile([char, tile[, ...]])`

  Like above, but also allows overriding the `tile` property on ad-hoc basis.

- `painter:string(text[, ...])`

  Paints the string with `dfhack.pen.parse(cur_pen,...)`; returns *self*.

- `painter:key(keycode[, ...])`

  Paints the description of the keycode using `dfhack.pen.parse(cur_key_pen,...)`; returns *self*.

As noted above, all painting methods return *self*, in order to allow chaining them like this:

```
painter:pen(foo):seek(x,y):char(1):advance(1):string('bar')...
```

View class

This class is the common abstract base of both the stand-alone screens and common widgets to be used inside them. It defines the basic layout, rendering and event handling framework.

The class defines the following attributes:

visible Specifies that the view should be painted.

active Specifies that the view should receive events, if also visible.

view_id Specifies an identifier to easily identify the view among subviews. This is reserved for implementation of top-level views, and should not be used by widgets for their internal subviews.

It also always has the following fields:

subviews Contains a table of all subviews. The sequence part of the table is used for iteration. In addition, subviews are also indexed under their *view_id*, if any; see `addviews()` below.

These fields are computed by the layout process:

frame_parent_rect The ViewRect represeting the client area of the parent view.

frame_rect The `mkdims` rect of the outer frame in parent-local coordinates.

frame_body The ViewRect representing the body part of the View's own frame.

The class has the following methods:

- `view:addviews(list)`

  Adds the views in the list to the `subviews` sequence. If any of the views in the list have `view_id` attributes that don't conflict with existing keys in `subviews`, also stores them under the string keys. Finally, copies any non-conflicting string keys from the `subviews` tables of the listed views.

  Thus, doing something like this:

```
self:addviews{
    Panel{
        view_id = 'panel',
        subviews = {
            Label{ view_id = 'label' }
        }
    }
}
```

  Would make the label accessible as both `self.subviews.label` and `self.subviews.panel.subviews.label`.

- `view:getWindowSize()`

  Returns the dimensions of the `frame_body` rectangle.

- `view:getMousePos()`

  Returns the mouse *x,y* in coordinates local to the `frame_body` rectangle if it is within its clip area, or nothing otherwise.

- `view:updateLayout([parent_rect])`

  Recomputes layout of the view and its subviews. If no argument is given, re-uses the previous parent rect. The process goes as follows:

  1. Calls `preUpdateLayout(parent_rect)` via `invoke_before`.
  2. Uses `computeFrame(parent_rect)` to compute the desired frame.
  3. Calls `postComputeFrame(frame_body)` via `invoke_after`.
  4. Calls `updateSubviewLayout(frame_body)` to update children.
  5. Calls `postUpdateLayout(frame_body)` via `invoke_after`.

- `view:computeFrame(parent_rect)` *(for overriding)*

  Called by `updateLayout` in order to compute the frame rectangle(s). Should return the `mkdims` rectangle for the outer frame, and optionally also for the body frame. If only one rectangle is returned, it is used for both frames, and the margin becomes zero.

- `view:updateSubviewLayout(frame_body)`

  Calls `updateLayout` on all children.

- `view:render(painter)`

  Given the parent's painter, renders the view via the following process:

  1. Calls `onRenderFrame(painter, frame_rect)` to paint the outer frame.
  2. Creates a new painter using the `frame_body` rect.
  3. Calls `onRenderBody(new_painter)` to paint the client area.
  4. Calls `renderSubviews(new_painter)` to paint visible children.

- `view:renderSubviews(painter)`

  Calls `render` on all `visible` subviews in the order they appear in the `subviews` sequence.

- `view:onRenderFrame(painter, rect)` *(for overriding)*

  Called by `render` to paint the outer frame; by default does nothing.

- `view:onRenderBody(painter)` *(for overriding)*

  Called by `render` to paint the client area; by default does nothing.

- `view:onInput(keys)` *(for overriding)*

  Override this to handle events. By default directly calls `inputToSubviews`. Return a true value from this method to signal that the event has been handled and should not be passed on to more views.

- `view:inputToSubviews(keys)`

  Calls `onInput` on all visible active subviews, iterating the `subviews` sequence in *reverse order*, so that topmost subviews get events first. Returns *true* if any of the subviews handled the event.

Screen class

This is a View subclass intended for use as a stand-alone dialog or screen. It adds the following methods:

- `screen:isShown()`

  Returns *true* if the screen is currently in the game engine's display stack.

- `screen:isDismissed()`

  Returns *true* if the screen is dismissed.

- `screen:isActive()`

  Returns *true* if the screen is shown and not dismissed.

- `screen:invalidate()`

  Requests a repaint. Note that currently using it is not necessary, because repaints are constantly requested automatically, due to issues with native screens happening otherwise.

- `screen:renderParent()`

  Asks the parent native screen to render itself, or clears the screen if impossible.

- `screen:sendInputToParent(...)`

  Uses `simulateInput` to send keypresses to the native parent screen.

- `screen:show([parent])`

  Adds the screen to the display stack with the given screen as the parent; if parent is not specified, places this one one topmost. Before calling `dfhack.screen.show`, calls `self:onAboutToShow(parent)`.

- `screen:onAboutToShow(parent)` *(for overriding)*

  Called when `dfhack.screen.show` is about to be called.

- `screen:onShow()`

  Called by `dfhack.screen.show` once the screen is successfully shown.

- `screen:dismiss()`

  Dismisses the screen. A dismissed screen does not receive any more events or paint requests, but may remain in the display stack for a short time until the game removes it.

- `screen:onDismiss()` *(for overriding)*

  Called by `dfhack.screen.dismiss()`.

- `screen:onDestroy()` *(for overriding)*

  Called by the native code when the screen is fully destroyed and removed from the display stack. Place code that absolutely must be called whenever the screen is removed by any means here.

- `screen:onResize`, `screen:onRender`

  Defined as callbacks for native code.

FramedScreen class

A Screen subclass that paints a visible frame around its body. Most dialogs should inherit from this class.

A framed screen has the following attributes:

frame_style A table that defines a set of pens to draw various parts of the frame.

frame_title A string to display in the middle of the top of the frame.

frame_width Desired width of the client area. If *nil*, the screen will occupy the whole width.

frame_height Likewise, for height.

frame_inset The gap between the frame and the client area. Defaults to 0.

frame_background The pen to fill in the frame with. Defaults to CLEAR_PEN.

There are the following predefined frame style tables:

- `GREY_FRAME`

  A plain grey-colored frame.

- `BOUNDARY_FRAME`

  The same frame as used by the usual full-screen DF views, like dwarfmode.

- `GREY_LINE_FRAME`

  A frame consisting of grey lines, similar to the one used by titan announcements.

gui.widgets

This module implements some basic widgets based on the View infrastructure.

Widget class

Base of all the widgets. Inherits from View and has the following attributes:

- `frame = {...}`

  Specifies the constraints on the outer frame of the widget. If omitted, the widget will occupy the whole parent rectangle.

  The frame is specified as a table with the following possible fields:

  l gap between the left edges of the frame and the parent.

  t gap between the top edges of the frame and the parent.

  r gap between the right edges of the frame and the parent.

  b gap between the bottom edges of the frame and the parent.

  w maximum width of the frame.

  h maximum heigth of the frame.

  xalign X alignment of the frame.

  yalign Y alignment of the frame.

  First the `l,t,r,b` fields restrict the available area for placing the frame. If `w` and `h` are not specified or larger then the computed area, it becomes the frame. Otherwise the smaller frame is placed within the are based on the `xalign/yalign` fields. If the align hints are omitted, they are assumed to be 0, 1, or 0.5 based on which of the `l/r/t/b` fields are set.

- `frame_inset = {...}`

  Specifies the gap between the outer frame, and the client area. The attribute may be a simple integer value to specify a uniform inset, or a table with the following fields:

  l left margin.

  t top margin.

  r right margin.

  b bottom margin.

  x left/right margin, if `l` and/or `r` are omitted.

  y top/bottom margin, if `t` and/or `b` are omitted.

- `frame_background = pen`

  The pen to fill the outer frame with. Defaults to no fill.

Panel class

Inherits from Widget, and intended for grouping a number of subviews.

Has attributes:

- `subviews = {}`

  Used to initialize the subview list in the constructor.

- `on_render = function(painter)`

  Called from `onRenderBody`.

Pages class

Subclass of Panel; keeps exactly one child visible.

- `Pages{ ..., selected = ... }`

  Specifies which child to select initially; defaults to the first one.

- `pages:getSelected()`

  Returns the selected *index, child*.

- `pages:setSelected(index)`

  Selects the specified child, hiding the previous selected one. It is permitted to use the subview object, or its `view_id` as index.

EditField class

Subclass of Widget; implements a simple edit field.

Attributes:

text The current contents of the field.

text_pen The pen to draw the text with.

on_char Input validation callback; used as `on_char(new_char,text)`. If it returns false, the character is ignored.

on_change Change notification callback; used as `on_change(new_text,old_text)`.

on_submit Enter key callback; if set the field will handle the key and call `on_submit(text)`.

Label class

This Widget subclass implements flowing semi-static text.

It has the following attributes:

text_pen Specifies the pen for active text.

text_dpen Specifies the pen for disabled text.

disabled Boolean or a callback; if true, the label is disabled.

enabled Boolean or a callback; if false, the label is disabled.

auto_height Sets self.frame.h from the text height.

auto_width Sets self.frame.w from the text width.

The text itself is represented as a complex structure, and passed to the object via the `text` argument of the constructor, or via the `setText` method, as one of:

- A simple string, possibly containing newlines.
- A sequence of tokens.

Every token in the sequence in turn may be either a string, possibly containing newlines, or a table with the following possible fields:

- `token.text = ...`

  Specifies the main text content of a token, and may be a string, or a callback returning a string.

- `token.gap = ...`

  Specifies the number of character positions to advance on the line before rendering the token.

- `token.tile = pen`

  Specifies a pen to paint as one tile before the main part of the token.

- `token.width = ...`

  If specified either as a value or a callback, the text field is padded or truncated to the specified number.

- `token.pad_char = '?'`

  If specified together with `width`, the padding area is filled with this character instead of just being skipped over.

- `token.key = '...'`

  Specifies the keycode associated with the token. The string description of the key binding is added to the text content of the token.

- `token.key_sep = '...'`

  Specifies the separator to place between the keybinding label produced by `token.key`, and the main text of the token. If the separator is '()', the token is formatted as `text..' ('..binding..')'`. Otherwise it is simply `binding..sep..text`.

- `token.enabled`, `token.disabled`

  Same as the attributes of the label itself, but applies only to the token.

- `token.pen`, `token.dpen`

  Specify the pen and disabled pen to be used for the token's text. The field may be either the pen itself, or a callback that returns it.

- `token.on_activate`

  If this field is not nil, and `token.key` is set, the token will actually respond to that key binding unless disabled, and call this callback. Eventually this may be extended with mouse click support.

- `token.id`

  Specifies a unique identifier for the token.

- `token.line`, `token.x1`, `token.x2`

  Reserved for internal use.

The Label widget implements the following methods:

- `label:setText(new_text)`

  Replaces the text currently contained in the widget.

- `label:itemById(id)`

  Finds a token by its `id` field.

- `label:getTextHeight()`

  Computes the height of the text.

- `label:getTextWidth()`

  Computes the width of the text.

List class

The List widget implements a simple list with paging.

It has the following attributes:

text_pen Specifies the pen for deselected list entries.

cursor_pen Specifies the pen for the selected entry.

inactive_pen If specified, used for the cursor when the widget is not active.

icon_pen Default pen for icons.

on_select Selection change callback; called as `on_select(index,choice)`. This is also called with *nil* arguments if `setChoices` is called with an empty list.

on_submit Enter key callback; if specified, the list reacts to the key and calls it as `on_submit(index,choice)`.

on_submit2 Shift-Enter key callback; if specified, the list reacts to the key and calls it as `on_submit2(index,choice)`.

row_height Height of every row in text lines.

icon_width If not *nil*, the specified number of character columns are reserved to the left of the list item for the icons.

scroll_keys Specifies which keys the list should react to as a table.

Every list item may be specified either as a string, or as a lua table with the following fields:

text Specifies the label text in the same format as the Label text.

caption, [1] Deprecated legacy aliases for **text**.

text_* Reserved for internal use.

key Specifies a keybinding that acts as a shortcut for the specified item.

icon Specifies an icon string, or a pen to paint a single character. May be a callback.

icon_pen When the icon is a string, used to paint it.

The list supports the following methods:

- `List{ ..., choices = ..., selected = ...  }`

  Same as calling `setChoices` after construction.

- `list:setChoices(choices[, selected])`

  Replaces the list of choices, possibly also setting the currently selected index.

- `list:setSelected(selected)`

  Sets the currently selected index. Returns the index after validation.

- `list:getChoices()`

  Returns the list of choices.

- `list:getSelected()`

  Returns the selected *index, choice*, or nothing if the list is empty.

- `list:getContentWidth()`

  Returns the minimal width to draw all choices without clipping.

- `list:getContentHeight()`

  Returns the minimal width to draw all choices without scrolling.

- `list:submit()`

  Call the `on_submit` callback, as if the Enter key was handled.

- `list:submit2()`

  Call the `on_submit2` callback, as if the Shift-Enter key was handled.

FilteredList class

This widget combines List, EditField and Label into a combo-box like construction that allows filtering the list by subwords of its items.

In addition to passing through all attributes supported by List, it supports:

edit_pen If specified, used instead of `cursor_pen` for the edit field.

edit_below If true, the edit field is placed below the list instead of above.

not_found_label Specifies the text of the label shown when no items match the filter.

The list choices may include the following attributes:

search_key If specified, used instead of **text** to match against the filter.

The widget implements:

- `list:setChoices(choices[, selected])`

  Resets the filter, and passes through to the inner list.

- `list:getChoices()`

  Returns the list of *all* choices.

- `list:getFilter()`

  Returns the current filter string, and the *filtered* list of choices.

- `list:setFilter(filter[,pos])`

  Sets the new filter string, filters the list, and selects the item at index `pos` in the *unfiltered* list if possible.

- `list:canSubmit()`

  Checks if there are currently any choices in the filtered list.

- `list:getSelected()`, `list:getContentWidth()`, `list:getContentHeight()`, `list:submit()`

  Same as with an ordinary list.

4.3.5 Plugins

DFHack plugins may export native functions and events to lua contexts. They are automatically imported by `mkmodule('plugins.<name>')`; this means that a lua module file is still necessary for `require` to read.

The following plugins have lua support.

burrows

Implements extended burrow manipulations.

Events:

- `onBurrowRename.foo = function(burrow)`

  Emitted when a burrow might have been renamed either through the game UI, or `renameBurrow()`.

- `onDigComplete.foo = function(job_type,pos,old_tiletype,new_tiletype,worker)`

  Emitted when a tile might have been dug out. Only tracked if the auto-growing burrows feature is enabled.

Native functions:

- `renameBurrow(burrow,name)`

  Renames the burrow, emitting `onBurrowRename` and updating auto-grow state properly.

- `findByName(burrow,name)`

  Finds a burrow by name, using the same rules as the plugin command line interface. Namely, trailing `'+'` characters marking auto-grow burrows are ignored.

- `copyUnits(target,source,enable)`

  Applies units from `source` burrow to `target`. The `enable` parameter specifies if they are to be added or removed.

- `copyTiles(target,source,enable)`

  Applies tiles from `source` burrow to `target`. The `enable` parameter specifies if they are to be added or removed.

- `setTilesByKeyword(target,keyword,enable)`

  Adds or removes tiles matching a predefined keyword. The keyword set is the same as used by the command line.

The lua module file also re-exports functions from `dfhack.burrows`.

sort

Does not export any native functions as of now. Instead, it calls lua code to perform the actual ordering of list items.

Eventful

This plugin exports some events to lua thus allowing to run lua functions on DF world events.

List of events

1. `onReactionComplete(reaction,reaction_product,unit,input_items,input_reagents,output_it`

   Auto activates if detects reactions starting with `LUA_HOOK_`. Is called when reaction finishes.

2. `onItemContaminateWound(item,unit,wound,number1,number2)`

   Is called when item tries to contaminate wound (e.g. stuck in).

3. `onProjItemCheckMovement(projectile)`

   Is called when projectile moves.

4. `onProjItemCheckImpact(projectile,somebool)`

   Is called when projectile hits something.

5. `onProjUnitCheckMovement(projectile)`

   Is called when projectile moves.

6. `onProjUnitCheckImpact(projectile,somebool)`

   Is called when projectile hits something.

7. `onWorkshopFillSidebarMenu(workshop,callnative)`

   Is called when viewing a workshop in 'q' mode, to populate reactions, useful for custom viewscreens for shops.

8. `postWorkshopFillSidebarMenu(workshop)`

   Is called after calling (or not) native fillSidebarMenu(). Useful for job button tweaking (e.g. adding custom reactions)

Events from EventManager

These events are straight from EventManager module. Each of them first needs to be enabled. See functions for more info. If you register a listener before the game is loaded, be aware that no events will be triggered immediately after loading, so you might need to add another event listener for when the game first loads in some cases.

1. `onBuildingCreatedDestroyed(building_id)`

   Gets called when building is created or destroyed.

2. `onConstructionCreatedDestroyed(building_id)`

   Gets called when construction is created or destroyed.

3. `onJobInitiated(job)`

   Gets called when job is issued.

4. `onJobCompleted(job)`

   Gets called when job is finished. The job that is passed to this function is a copy. Requires a frequency of 0 in order to distinguish between workshop jobs that were cancelled by the user and workshop jobs that completed successfully.

5. `onUnitDeath(unit_id)`

   Gets called on unit death.

6. `onItemCreated(item_id)`

   Gets called when item is created (except due to traders, migrants, invaders and spider webs).

7. `onSyndrome(unit_id,syndrome_index)`

   Gets called when new syndrome appears on a unit.

8. `onInvasion(invasion_id)`

   Gets called when new invasion happens.

9. `onInventoryChange(unit_id,item_id,old_equip,new_equip)`

   Gets called when someone picks up an item, puts one down, or changes the way they are holding it. If an item is picked up, old_equip will be null. If an item is dropped, new_equip will be null. If an item is re-equipped in a new way, then neither will be null. You absolutely must NOT alter either old_equip or new_equip or you might break other plugins.

10. `onReport(reportId)`

    Gets called when a report happens. This happens more often than you probably think, even if it doesn't show up in the announcements.

11. `onUnitAttack(attackerId, defenderId, woundId)`

    Called when a unit wounds another with a weapon. Is NOT called if blocked, dodged, deflected, or parried.

12. `onUnload()`

    A convenience event in case you don't want to register for every onStateChange event.

13. `onInteraction(attackVerb, defendVerb, attackerId, defenderId, attackReportId, defendReportId)`

    Called when a unit uses an interaction on another.

Functions

1. `registerReaction(reaction_name,callback)`

   Simplified way of using onReactionComplete; the callback is function (same params as event).

2. `removeNative(shop_name)`

   Removes native choice list from the building.

3. `addReactionToShop(reaction_name,shop_name)`

   Add a custom reaction to the building.

4. `enableEvent(evType,frequency)`

   Enable event checking for EventManager events. For event types use `eventType` table. Note that different types of events require different frequencies to be effective. The frequency is how many ticks EventManager will wait before checking if that type of event has happened. If multiple scripts or plugins use the same event type, the smallest frequency is the one that is used, so you might get events triggered more often than the frequency you use here.

5. `registerSidebar(shop_name,callback)`

   Enable callback when sidebar for `shop_name` is drawn. Usefull for custom workshop views e.g. using gui.dwarfmode lib. Also accepts a `class` instead of function as callback. Best used with `gui.dwarfmode` class `WorkshopOverlay`.

Examples

Spawn dragon breath on each item attempt to contaminate wound:

```
b=require "plugins.eventful"
b.onItemContaminateWound.one=function(item,unit,un_wound,x,y)
    local flw=dfhack.maps.spawnFlow(unit.pos,6,0,0,50000)
end
```

Reaction complete example:

```
b=require "plugins.eventful"

b.registerReaction("LUA_HOOK_LAY_BOMB",function(reaction,unit,in_items,in_reag,out_items,call_native)
  local pos=copyall(unit.pos)
  -- spawn dragonbreath after 100 ticks
  dfhack.timeout(100,"ticks",function() dfhack.maps.spawnFlow(pos,6,0,0,50000) end)
  --do not call real item creation code
  call_native.value=false
end)
```

Grenade example:

```
b=require "plugins.eventful"
b.onProjItemCheckImpact.one=function(projectile)
  -- you can check if projectile.item e.g. has correct material
  dfhack.maps.spawnFlow(projectile.cur_pos,6,0,0,50000)
end
```

Integrated tannery:

```
b=require "plugins.eventful"
b.addReactionToShop("TAN_A_HIDE","LEATHERWORKS")
```

Building-hacks

This plugin overwrites some methods in workshop df class so that mechanical workshops are possible. Although plugin export a function it's recommended to use lua decorated function.

Functions

`registerBuilding(table)` where table must contain name, as a workshop raw name, the rest are optional:

name custom workshop id e.g. `SOAPMAKER`

> **Note:** this is the only mandatory field.

fix_impassible if true make impassible tiles impassible to liquids too

consume how much machine power is needed to work. Disables reactions if not supplied enough and `needs_power==1`

produce how much machine power is produced.

needs_power if produced in network < consumed stop working, default true

gears a table or `{x=?,y=?}` of connection points for machines.

action a table of number (how much ticks to skip) and a function which gets called on shop update

animate a table of frames which can be a table of:

1. tables of 4 numbers `{tile,fore,back,bright}` OR
2. empty table (tile not modified) OR
3. `{x=<number> y=<number> + 4 numbers like in first case}`, this generates full frame useful for animations that change little (1-2 tiles)

canBeRoomSubset a flag if this building can be counted in room. 1 means it can, 0 means it can't and -1 default building behaviour

auto_gears a flag that automatically fills up gears and animate. It looks over building definition for gear icons and maps them.

Animate table also might contain:

frameLength how many ticks does one frame take OR

isMechanical a bool that says to try to match to mechanical system (i.e. how gears are turning)

`getPower(building)` returns two number - produced and consumed power if building can be modified and returns nothing otherwise

`setPower(building,produced,consumed)` sets current productiona and consumption for a building.

Examples

Simple mechanical workshop:

```
require('plugins.building-hacks').registerBuilding{name="BONE_GRINDER",
  consume=15,
  gears={x=0,y=0}, --connection point
  animate={
    isMechanical=true, --animate the same conn. point as vanilla gear
    frames={
    {{x=0,y=0,42,7,0,0}}, --first frame, 1 changed tile
    {{x=0,y=0,15,7,0,0}} -- second frame, same
```

```
    }
  }
```

Or with auto_gears:

```
require('plugins.building-hacks').registerBuilding{name="BONE_GRINDER",
  consume=15,
  auto_gears=true
  }
```

Luasocket

A way to access csocket from lua. The usage is made similar to luasocket in vanilla lua distributions. Currently only subset of functions exist and only tcp mode is implemented.

Socket class

This is a base class for `client` and `server` sockets. You can not create it - it's like a virtual base class in c++.

- `socket:close()`

  Closes the connection.

- `socket:setTimeout(sec,msec)`

  Sets the operation timeout for this socket. It's possible to set timeout to 0. Then it performs like a non-blocking socket.

Client class

Client is a connection socket to a server. You can get this object either from `tcp:connect(address,port)` or from `server:accept()`. It's a subclass of `socket`.

- `client:receive(pattern)`

  Receives data. Pattern is one of:

  `*l` read one line (default, if pattern is *nil*)

  <number> read specified number of bytes

  `*a` read all available data

- `client:send(data)`

  Sends data. Data is a string.

Server class

Server is a socket that is waiting for clients. You can get this object from `tcp:bind(address,port)`.

- `server:accept()`

  Accepts an incoming connection if it exists. Returns a `client` object representing that socket.

Tcp class

A class with all the tcp functionality.

- `tcp:bind(address,port)`

  Starts listening on that port for incoming connections. Returns `server` object.

- `tcp:connect(address,port)`

  Tries connecting to that address and port. Returns `client` object.

4.3.6 Scripts

Any files with the .lua extension placed into `hack/scripts/*` are automatically used by the DFHack core as commands. The matching command name consists of the name of the file without the extension. First DFHack searches for the script in the `<save_folder>/raw/scripts/` folder. If it is not found there, it searches in the `<DF>/raw/scripts/` folder. If it is not there, it searches in `<DF>/hack/scripts/`. If it is not there, it gives up.

If the first line of the script is a one-line comment, it is used by the built-in `ls` and `help` commands. Such a comment is required for every script in the official DFHack repository.

Note: Scripts placed in subdirectories still can be accessed, but do not clutter the *ls* command list (unless `ls -a`; thus it is preferred for obscure developer-oriented scripts and scripts used by tools. When calling such scripts, always use '/' as the separator for directories, e.g. `devel/lua-example`.

Scripts are re-read from disk if they have changed since the last time they were read. Global variable values persist in memory between calls, unless the file has changed. Every script gets its own separate environment for global variables.

Arguments are passed in to the scripts via the **...** built-in quasi-variable; when the script is called by the DFHack core, they are all guaranteed to be non-nil strings.

DFHack core invokes the scripts in the *core context* (see above); however it is possible to call them from any lua code (including from other scripts) in any context, via the same function the core uses:

- `dfhack.run_script(name[,args...])`

  Run a lua script in hack/scripts/, as if it was started from dfhack command-line. The `name` argument should be the name stem, as would be used on the command line.

Note that this function lets errors propagate to the caller.

- `dfhack.script_environment(name)`

  Run an Lua script and return its environment. This command allows you to use scripts like modules for increased portability. It is highly recommended that if you are a modder you put your custom modules in `raw/scripts` and use `script_environment` instead of `require` so that saves with your mod installed will be self-contained and can be transferred to people who do have DFHack but do not have your mod installed.

  You can say `dfhack.script_environment('add-thought').addEmotionToUnit([arguments go here])` and it will have the desired effect. It will call the script in question with the global `moduleMode` set to `true` so that the script can return early. This is useful because if the script is called from the console it should deal with its console arguments and if it is called by `script_environment` it

should only create its global functions and return. You can also access global variables with, for example `print(dfhack.script_environment('add-thought').validArgs)`

The function `script_environment` is fast enough that it is recommended that you not store its result in a nonlocal variable, because your script might need to load a different version of that script if the save is unloaded and a save with a different mod that overrides the same script with a slightly different functionality is loaded. This will not be an issue in most cases.

This function also permits circular dependencies of scripts.

- `dfhack.reqscript(name)` or `reqscript(name)`

  Nearly identical to script_environment() but requires scripts being loaded to include a line similar to:

```
--@ module = true
```

  This is intended to only allow scripts that take appropriate action when used as a module to be loaded.

Enabling and disabling scripts

Scripts can choose to recognize the built-in `enable` and `disable` commands by including the following line anywhere in their file:

```
--@ enable = true
```

When the `enable` and `disable` commands are invoked, a `dfhack_flags` table will be passed to the script with the following fields set:

- `enable`: Always true if the script is being enabled *or* disabled
- `enable_state`: True if the script is being enabled, false otherwise

Save init script

If a save directory contains a file called `raw/init.lua`, it is automatically loaded and executed every time the save is loaded. The same applies to any files called `raw/init.d/*.lua`. Every such script can define the following functions to be called by dfhack:

- `function onStateChange(op) ... end`

  Automatically called from the regular onStateChange event as long as the save is still loaded. This avoids the need to install a hook into the global `dfhack.onStateChange` table, with associated cleanup concerns.

- `function onUnload() ... end`

  Called when the save containing the script is unloaded. This function should clean up any global hooks installed by the script. Note that when this is called, the world is already completely unloaded.

Within the init script, the path to the save directory is available as `SAVE_PATH`.

4.4 Data Structure Definition Syntax

Contents

This document documents the XML syntax used to define DF data structures for use in dfhack.

4.4.1 General Background

Originally dfhack used a file called `Memory.xml` to describe data structures of the game. It explicitly listed addresses of known global variables, and offsets within structures to fields, not unlike the ini files used by Dwarf Therapist.

This format is a good choice when only a small number of fields and objects need to be accessed, and allows a program to work with many different versions of DF, provided that the relevant fields and objects work in the same way.

However, as the number of known fields and objects grow, maintaining the explicit offset lists quickly becomes difficult and error prone. Also, even when almost all fields of a structure become known, the format fails to represent and exploit their relative position, which in practice is actually more stable than the specific offset values.

This format instead represents data structure layout purely via listing all fields in the correct order, exactly like a structure definition does in the C++ language itself; in fact, these XML definitions are translated into C++ headers in a

mostly straightforward way (the more tricky bits are things like correctly processing circular references, or generating metadata for lua). There is still a file with numeric data, but it only contains absolute addresses of global objects.

As a downside, dfhack now needs to be recompiled every time layout of some data structure changes; on the other hand, accessing DF structures from C++ plugins now has no overhead compared with DF's own code. Also, practice shows that the more fields are known in a structure, the easier it is to spot what exactly has changed, and fix the exact area.

4.4.2 XML file format

All XML files use `<data-definition>` as their root tag.

They should be indented using 4 spaces per level, without tabs.

Unless noted otherwise, all non-root tags allow using a *comment* attribute, or a `<comment>...</comment>` subtag. It may be used to include a comment string that can be used by tools processing the xml.

Excluding content of tags like `<comment>` or `<code-helper>`, all plain text inside tag bodies is ignored and may be freely used instead of XML comments.

NOTE: Using XML tags and/or attributes not defined in this document is not allowed.

Enum type definition

Global enum types are defined as follows:

```
<enum-type type-name='name' [base-type='int32_t']>
    <enum-item [name='key1'] [value='0']/>
    <enum-item [name='key2'] [value='1']/>
    ...
</enum-type>
```

Every enum has an integer base type, which defaults to *int32_t* if omitted.

Like in C++, enum items may either explicitly specify an integer value, or rely on auto-increment behavior.

NOTE: Due to a codegen limitation, specifying value on any item other than the first one prevents using the attribute feature described below.

As in most cases, the *name* attribute may be omitted if unknown; the code generator would produce a random identifier to satisfy C++ language requirements.

Enum item attributes

The XML syntax allows associating attributes with enum items, thus embedding lookup tables for use in C++ or lua code.

Every attribute must be declared at the top level of the enum:

```
<enum-attr name='attr'
          [type-name='primitive-or-enum']
          [default-value='...']
          [use-key-name='true/false']
          [is-list='true/false']/>
```

The declaration allows specifying a numeric, or other enum type for the attribute, overriding the default `const char*` string type.

An explicit default value may also be specified; otherwise the attribute defaults to NULL or 0. If `use-key-name` is *true*, the corresponding `enum-item`'s *name* is used as the default value.

Alternatively, an attribute may be declared to be a list, instead of a scalar. In this case, the default is an empty list.

NOTE: Attribute name `'key'` is reserved for a built-in string attribute representing the enum item key.

For every declared attribute, every enum-item tag may contain an attribute value definition:

```
<enum-item name='key'>
    <item-attr name='attr' value='...'/>
    ...
</enum-item>
```

For list attributes, multiple `item-attr` entries may be used to define the list contents.

Bitfield type definition

Global bitfield types are defined as follows:

```
<bitfield-type type-name='name' [base-type='uint32_t']>
    <flag-bit [name='bit1'] [count='1'] [type-name='enum']/>
    <flag-bit [name='bit2'] [count='1'] [type-name='enum']/>
    ...
</bitfield-type>
```

Like enums, bitfields have an integer base type, which defaults to *uint32_t*. The total number of bits in the bitfield must not exceed the base type size.

A bitfield item may be defined to occupy multiple bits via the *count* attribute. It also may have an enum type; due to compiler limitations, the base-type of the enum must be exactly the same as the bitfield itself.

Structure type definition

Structures without virtual methods are defined as follows:

```
<struct-type type-name='name'
             [is-union='true/false']
             [inherits-from='struct_type']
             [instance-vector='expr']
             [key-field='identifier']>
    ...
    fields
    ...
</struct-type>
```

The *instance-vector* attribute may be used to specify a global vector that canonically contains all instances of the structure. Code generation uses it to produce a `find` static method. If *key-field* is specified, this method uses binary search by the referred field; otherwise it just indexes the vector with its integer argument.

Common field properties

All fields support the following attributes:

`name` Specifies the identifier naming the field.

This attribute may be omitted, in which case the code generator produces a random identifier. As follows from the word random, such identifiers aren't stable, and shouldn't be used to access the field.

`init-value` Specifies the value that should be assigned to the field by the constructor. By default the following values are used:

- For enums: the first element of the enum.
- For signed integer fields with `ref-target` or `refers-to`: -1.
- For other numeric fields, pointers and bitfields: 0.

`offset`, `size`, `alignment` Specifies the offset, size and alignment in bytes.

WARNING: Although allowed for any field by the XML syntax, and supported by the lisp GUI tool, code generation will fail with these attributes except in cases specifically shown below.

With the above caveat, `size` and `alignment` may also be used on the `struct-type` tag itself.

Primitive fields

Primitive fields can be classified as following:

1. Unmarked area:

```
<padding name='id' size='bytes' [alignment='1/2/4'] .../>
```

   This tag defines an area of raw bytes with unknown contents.

2. Numbers:

```
<int32_t name='id'.../>
```

   Supported number types are: `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, `int32_t`, `uint32_t`, `int64_t`, `uint64_t`, `s-float` (single float), `d-float` (double float).

3. Boolean:

```
<bool name='id'.../>
```

4. String:

```
<static-string name='id' size='bytes'.../>
<ptr-string name='id'.../>
<stl-string name='id'.../>
```

   These tags correspond to `char[bytes]`, `char*`, and `std::string`.

4. File Stream:

```
<stl-fstream name='id'/>
```

   This is not really a primitive type, but classified as such since it is treated as a predefined opaque object (a-la padding).

Primitives support the following attributes:

`refers-to='expr'`

Specifies a GUI hyperlink to an object returned by an arbitrary expression.

The expression receives the value of the field as `$`, and the reference to the field as `$$`.

`ref-target='type'`

Specifies a hyperlink to an instance of *type*, identified by the value of the field. The instance is retrieved via *instance-vector* and *key-field*, or a `<code-helper name='find-instance'>` in the target type definition.

`aux-value='expr'`

Specifies an additional value for use in the *find-instance* code helper.

Unlike *refers-to*, the expression receives the **reference** to the field as `$`, and a reference to the containing structure as `$$`; i.e. the arguments are shifted one step toward parent. This is because the value of the field is already implicitly passed to *find-instance*.

The *find-instance* helper receives the field value as `$`, and aux-value as `$$`.

Substructure fields

Nested structures are defined via the `compound` tag:

```
<compound name='id' type-name='struct_type'/>

<compound [name='id'] [is-union='true/false'] [key-field='id']>
    ...
    field
    ...
</compound>
```

As seen above, a nested structure may either use a global type defined elsewhere, or define an ad-hoc structure in-place. In the in-place case, omitting *name* has a special meaning of defining an anonymous nested struct or union.

Enum fields

Fields of enum types are defined as follows:

```
<enum name='id' type-name='enum_type' [base-type='int32_t']/>

<enum name='id' [base-type='int32_t']>
    <enum-item name='key1'.../>
    ...
</enum>
```

Like with substructures, enums may be either referenced globals, or ad-hoc definitions.

In the former case, when *base-type* of the field and the enum differ, a special wrapper is added to coerce the size, or, if impossible, the enum type is completely replaced with the *base-type*. The net effect is that the field *always* has the expected size and alignment.

If no *base-type* is specified on the field, the one in the global type definition has complete precedence. This is not recommended.

Nested bitfields

Ad-hoc bitfields are defined as follows:

```
<bitfield name='id' [base-type='uint32_t']>
    <flag-bit name='key1'.../>
    ...
</bitfield>
```

In order to reference a global type, use `<compound>`.

Container fields

A number of tags fall under the 'container' abstraction. The common element is that the fields they define reference objects of another type. This includes things like pointers, arrays or vectors.

Abstract container The basic syntactic property of a container is that it requires exactly one nested field tag in order to specify the contained item:

```
<container>
    <field .../>
</container>
```

NOTE: The `container` tag is used here as a placeholder for any real tag following the container syntax.

For convenience, the following automatic rewrite rules are applied:

1. The `type-name` attribute:

```
<container type-name='foo' .../>
```

   is rewritten into:

```
<container ...>
    <compound type-name='foo' .../>
</container>
```

   or, if *foo* is a primitive type:

```
<container ...>
    <foo .../>
</container>
```

2. The `pointer-type` attribute:

```
<container pointer-type='foo' .../>
```

   is rewritten into:

```
<container ...>
    <pointer type-name='foo' .../>
</container>
```

3. Multiple nested fields:

```
<container ...>
    <field1 .../>
    <field2 .../>
</container>
```

   are aggregated together:

```
<container ...>
    <compound ...>
        <field1 .../>
        <field2 .../>
    </compound>
</container>
```

4. If no item is specified, `padding` is assumed:

```
    <container>
        <padding size='4'/>
    </container>
```

NOTE: These rules are mutually exclusive, and it is an error to specify both of the attributes (unless it is `type-name='pointer'`), or combine nested fields with any of them.

When the above rewrites are applied and result in creation of a new tag, the following attributes are copied to it from the container tag, if applicable: `key-field`, `refers-to`, `ref-target`, `aux-value`. They otherwise have no effect on the container itself.

This means that:

```
<container pointer-type='int32_t' ref-target='foo'/>
```

eventually rewrites to:

```
<container pointer-type='int32_t' ref-target='foo'>
    <pointer type-name='int32_t' ref-target='foo'>
        <int32_t ref-target='foo'/>
    </pointer>
</container>
```

Abstract containers allow the following attributes:

`has-bad-pointers='true'`

Tells the GUI tool to ignore this field in some of its memory scans, because this container may contain invalid pointers, which can confuse the analysis code.

Pointer fields As seen above, the `pointer` tag is a subtype of abstract container.

If the pointer refers to an array of objects, instead of one instance, the *is-array* attribute should be used:

<pointer type-name='foo' is-array='true'/>

Currently this attribute is ignored by C++ code generation, but the GUI tool properly displays such fields as arrays.

Abstract sequence Containers that actually contain a sequence of objects support these additional attributes:

`index-refers-to='expr'`

Specifies a GUI hyperlink from any item in the container to the object returned by the expression.

The expression receives the index of the item in the container as `$`, and a reference to the container as `$$`.

`index-enum='enum_type'`

Associates an enum with the indices of the container. The GUI tries to use enum item names instead of numbers when displaying the items, and lua may allow using strings as indices.

Standard containers `<static-array name='id' count='123' .../>`

Defines a simple C++ array of the specified length.

`<stl-vector name='id'.../>`

Defines an `std::vector<item>` field.

`<stl-deque name='id'.../>`

Defines an `std::deque<item>` field.

`<stl-set name='id'.../>`

Defines an `std::set<item>` field.

`<stl-bit-vector name='id'.../>`

Defines an `std::vector<bool>` field.

STL defines `vector<bool>` as a special type that actually contains bits. These XML definitions use a separate tag for it; `<stl-vector type-name='bool'/>` is rendered into C++ as `vector<char>`.

DF-specific containers These are defined in df-code.lisp:

`<df-flagarray name='id' index-enum='enum'/>`

Defines a `BitArray<enum>` field.

`<df-static-flagarray name='id' index-enum='enum' count='numbytes'/>`

Defines a `StaticBitArray<numbytes,enum>` field.

`<df-array name='id' .../>`

Defines a `DfArray<item>` field.

`<df-linked-list name='id' type-name='foo_link'/>`

Defines an ad-hoc DF-style linked list. In C++ actually equivalent to:

```
<compound type-name='foo_link'/>
```

but allows the GUI to display it as a list.

Class type definition

In the context of these XML definitions, class denotes types with virtual methods:

```
<class-type type-name='name'
           [inherits-from='class_type']
           [original-name='vtable_name']
           ...>
    ...
    fields
    ...
    <virtual-methods>
        ...
        vmethods
        ...
    </virtual-methods>
</class-type>
```

Classes are generally the same as `<struct-type>`, including support for *instance-vector*. Unlike `<struct-type>` however, they don't allow `is-union='true'`.

There may only be one table of virtual methods per class-type. In subclasses it should only contain items added to the table of the superclass.

Virtual method definition

Virtual method definitions are placed within the `<virtual-methods>` section of a class type. No other tag may be placed within that section, including *comment*.

A virtual destructor is defined as follows:

```
<vmethod is-destructor='true'/>
```

Ordinary virtual methods use the following syntax:

```
<vmethod [name='id'] [ret-type='type']>
    [<ret-type .../>]
    <field1.../>
    <field2.../>
    ...
</vmethod>
```

The return type may be specified either as an attribute, or via a `ret-type` sub-tag. The subtag syntax follows the abstract container model outlined above. The attribute is exactly equivalent to `<ret-type type-name='type'/>` as subtag. If the return type is completely omitted, it is taken to be void.

Ordinary field definition tags within the vmethod tag are treated as method parameters.

If the *name* attribute is omitted, the vmethod is named randomly and made protected, so that calling it is impossible. This is the intended way of providing placeholders for completely unknown slots in the vtable.

Global object definition

Global objects are global pointers that are initialized from symbols.xml at runtime. Therefore, the tag itself is identical in syntax to `<pointer>`, except that it doesn't allow *is-array*:

```
<global-object name='id' type-name='...'/>

<global-object name='id'>
    <field.../>
</global-object>
```

C++ generation places them in the `df::global` namespace.

The *offset* attribute of the `global-object` tag represents the absolute address. As noted above, it may only be used in files intended for the GUI.

Symbol table definition

Symbol tables are defined in symbols.xml and loaded at runtime. They define locations of global objects and virtual tables.

The definition syntax is as follows:

```
<symbol-table name='...' os-type='...'>
    <md5-hash value='...'/>
    <binary-timestamp value='0x...'/>
    ...

    <global-address name='...' [value='0x...']/>
    ...
```

```
    <vtable-address name='...' [value='0x...']/>
    ...
</symbol-table>
```

The *name* attribute specifies an unique name of the symbol table. *os-type* specifies the applicable OS type, and must be one of `windows`, `linux`, `darwin`.

The `<md5-hash>` tag specifies the MD5 hash that is used to match the executable on Linux and OS/X. It will be ignored if used in a windows symbol table. Likewise, `<binary-timestamp>` is valid only for matching EXE files. A symbol table may contain multiple tags in order to match several executables; this is especially useful with MD5 hashes, which change with patching.

Global object addresses are specified with `<global-address>` tags. Virtual method table addresses may be pre-initialized with `<vtable-address>` tags.

It is allowed to specify addresses for objects and vtables that are otherwise not defined. Obviously, such values can only be used by directly quering the VersionInfo object in dfhack.

4.4.3 Lisp Integration

This XML file format was designed together with the `cl-linux-debug` Lisp tool, and has a number of aspects that closely integrate with its internals.

For instance, when loaded by that tool, all XML tags are converted directly into instances of classes that exactly match the name of the tag, and when the documentation above mentions expressions, that refers to Lisp expressions within the context of that library.

Reference expressions

In order to facilitate compact representation for long chains of dereferences that are commonly required when dealing with the data structures, `cl-linux-debug` defines a reader macro (i.e. basically a parser plugin) that adds a custom syntax for them. This syntax is triggered by special characters `$` and `@`.

Expressions written in that syntax expand into nested chains of calls to two generic functions named `$` and `@`, which implement correspondingly r-value and l-value dereference of their first argument using the second.

Dereference syntax

The reader macro understands the following syntactic patterns:

- `@`, `$`, `$$`, `$$$`, ...

  Lone `@` and sequences of `$` are parsed just as the ordinary lisp parser would. This allows referring to the `$` and `@` functions, and using sequences of `$` characters as implicit argument names.

- `$foo`

  A case-sensitive identifier preceeded by the `$` character is interned in the `cl-linux-debug.field-names` package as-is, and returned as the parsing result. The identifier may consist of letters, numbers, and `-` or `_` characters.

  The symbol is exported from its package and defined as a symbol macro expanding to `'$foo`, and thus behaves as a case-sensitive keyword (which however can be used as a lexical variable name). All field & type names and other identifiers in the XML definitions are loaded into memory as such symbols.

- `$foo:bar`

  This expands into `'($foo . $bar)`; such pairs of identifiers are used in some special contexts.

- `$foo.bar`, `@foo.bar`

  These expressions expand to correspondingly `($ foo '$bar)` and `(@ foo '$bar)`, representing thus r-value or l-value dereference of variable foo with literal key `$bar`.

  The name `foo` may only contain characters listed above, but is otherwise separated and parsed with the regular lisp parser.

- `$foo.*`, `$foo[*]`, `$foo.@`, `$foo[@]`, `@foo.*` ...

  These expand to `($ foo '*)`, `($ foo '@)` etc, thus effectively being a special case of dereference via a literal field name.

- `$foo[expr]`, `@foo[expr]`

  These expressions expand to correspondingly `($ foo expr)` and `(@ foo expr)`, and are useful for accessing array elements.

- `$foo.xxx[yyy].zzz`

  When dereference clauses are chained, they expand into nested calls to `$` and `@`, with the outermost depending on the first character, and all the inner ones being `@`.

  This example expands to: `($ (@ (@ foo '$xxx) yyy) '$zzz)`.

- `@$$foo.bar`, `$$$foo.bar`

  When the expression contains multiple initial `$` characters, all but the first one are prepended to the initial variable name.

  These examples expand to `(@ $$foo '$bar)` and `($ $$foo '$bar)`

  NOTE: Only the `$` character may be used in this way; `$@@foo.bar` is invalid.

- `$.foo`, `@$[bar]`, ...

  If the expression contains no initial identifier, the initial `$` sequence is used as one instead (after replacing `@` with `$` if necessary).

  These examples expand to: `($ $ '$foo)`, `(@ $$ bar)`.

  NOTE: Unlike the previous syntax pattern, this one uses *all* of the initial `$` and `@` characters.

- `$(func arg arg...).bar`

  If one initial `$` or `@` is immediately followed by parentheses, the contents of said parentheses are parsed as ordinary lisp code and used instead of the initial variable.

  The example expands to: `($ (func arg arg...) '$bar)`

- `@$(foo bar baz)`

  If an initial `@` is followed by one or more `$` characters and then parentheses, it is parsed as a lambda expression (anonymous function) with one argument consisting of those `$` characters.

  This example expands to: `(lambda ($) (foo bar baz))`

  NOTE: it is an error to use multiple initial `$` characters without `@` like this: `$$$(...)...`

Basic properties

As described above, dereference is actually implemented by two generic functions, `@` and `$`, which implement l-value and r-value dereference.

They are defined as such:

```
(defgeneric @ (obj key))
(defgeneric $ (obj key))
(defgeneric (setf $) (obj key))
```

Generally, l-value dereference returns an object that can be dereferenced further. R-value dereference with the same arguments may return the same object as l-value, or a simple scalar value, depending on the context.

Perhaps oppositely to the used terms, only the r-value dereference function may be used as the *syntactic* target of assignment; this is because you can't actually change the (conceptual) address of an object, only its contents; and l-value dereference returns an address. I.e. in C++ you can write `*a = ...`, but can't do `&a = ...`.

Any of the dereference functions may return a list to represent multiple possible values. Array objects often define `(@ foo '*)` to return all of the elements.

If either the obj or key argument of any of the functions is a list (including *NIL* as empty list), the functions loop over the list, and return a concatenation of the resulting return value lists. This allows using `$array.*.field` to get a list of all values of a field within array elements.

`($ obj t)` is defined as the *natural* value of an object; e.g. if obj is a reference to a numeric field, this will be its value. By default it is equal to the object itself. `($ obj key)` for any other key would fall back to `($ (@ obj key) t)` if no special handler for `$` with that key and object was defined.

Reference objects

The `cl-linux-debug` library represents typed pointers to objects in memory as objects of the `memory-object-ref` type.

Along with the expected address and type of the pointer, these objects also retain a history of dereferences that have led to this particular pointer, and define virtual fields to access this information. This history is similar to what the Back button in a browser uses.

All references by default have the following properties:

- `@ref.value`

  By default returns ref itself. May be hidden by struct fields and index-enum keys.

- `@ref[integer]`

  Returns a reference to address + size*int, i.e. offsets the pointer.

- `@ref.*`

  Returns a list of contained collection elements. By default empty.

- `@ref.@`

  Returns a list of subfields. By default empty.

- `@ref._parent`

  Returns the previous reference in the "back" chain.

- `@ref._global`

  Returns the nearest reference in the "back" chain that has a globally named type, i.e. one defined by a `struct-type`, `class-type` etc, and not by any nested substructures. This may return the ref itself.

- `@ref._upglobal`

  Exactly equivalent to `@ref._parent._global`.

- `$ref._address`

  Returns the numeric address embedded in the ref.

- `$ref._size`

  Returns the size of the object pointed to.

- `$ref._key`

  Returns the key that was used to get this ref from the parent. This is not guaranteed to be precisely accurate, but e.g. for array elements this will be the array index.

- `$ref._type`

  For globally named types, returns their type name.

Primitive types

Primitive types define the following methods:

- `$ref[t]`

  The natural value of a primitive field is the scalar non-reference value it contains.

  NOTE: When you write `$struct.field`, it will evaluate via `($ @struct.field t)`.

- `@ref.refers-to`, `@ref.ref-target`

  If the field has the relevant attributes, they can be dereferenced to retrieve the target objects.

Enums

Enum fields return their value as symbols, and allow access to attributes:

- `$ref[t]`

  Returns the symbol matching the value, unless there is none. May be assigned both as symbol or number.

- `$ref.attribute`

  If the enum has an attribute with that name, retrieves its value for the current value of the field.

Pointers

- `$ref[t]`, `@ref[t]`, `$ref._target`, `@ref._target`

  These all return the value of the pointer, i.e. a reference to the target object.

- `($ ref key)` -> `($ (@ ref t) key)`
- `(@ ref key)` -> `(@ (@ ref t) key)`

  All dereferences not explicitly supported are delegated to the target object. This means that for most properties pointers are completely transparent; notable exceptions are pointers to pointers, and pointers to primitive fields where you have to use e.g. `$struct.ptrfield.value`.

Compounds

- `@ref.field`, `@ref._fields.field`

  Returns a reference to the given field.

- `@ref.*`, `@ref.@`

  Returns a list of references to all fields. Note that if the object is both an implicit compound and a sequence, `@ref.*` will returns the sequence items as described below.

Sequences

- `@ref[int]`

  Returns a reference to the Nth item of the sequence.

- `@ref[symbol]`

  If the sequence has an `index-enum`, its items can be accessed by symbolic names.

- `@ref.*`

  Returns a list of all items of the sequence.

- `@ref._items`

  Returns the items of the sequence as a special lazy object, intended to optimize some things in the GUI.

- `@ref.index-refers-to[int]`

  If the sequence has the relevant attribute, returns the target for the given index.

- `$ref.count`

  Returns the number of items in the sequence.

- `$ref.has-items`

  Checks if the sequence has any items, and returns T or NIL.

Code helpers

The `<code-helper>` tag may be used to add lisp code fragments to the objects defined in the xml. The `refers-to`, `index-refers-to` and `ref-target` tags are also converted to code helpers internally, and you can use e.g. `<code-helper name='refers-to'>...</code-helper>` instead of the attribute if your expression is too long for it.

There are two features that can only be implemented via explicit `<code-helper>` tags:

- `<code-helper name='describe'> ... </code-helper>`

  This specifies a piece of code that is called to supply additional informational items for the rightmost column of the table in the GUI tool. The code should return a string, or a list of strings.

  As with `refers-to`, the code receives the value of the object as `$`, and the reference to the object in `$$` (i.e. `$` is equal to `$$[t]`).

  The `(describe-obj object)` function can be used to call the same describe mechanism on another object, e.g.:

```
<code-helper name='describe'> (describe-obj $.name) </code-helper>
```

- `<code-helper name='find-instance'> ... </code-helper>`

  If the `instance-vector` and `key-field` attributes are not descriptive enough to specify how to find an instance of the object by id, you can explicitly define this helper to be used by `ref-target` links elsewhere.

  It receives the value of the `ref-target` bearing field as `$`, and its `aux-value` as `$$`.

  Other than via `ref-target`, you can invoke this mechanism explicitly using the `(find-instance class key aux-key)` function, even from a `find-instance` helper for another type:

```
<code-helper name='find-instance'>$(find-instance $art_image_chunk $$).images[$]</code-helper>
```

  This finds an instance of the `art_image_chunk` type using the aux-value `$$`, and then returns an element of its `images` sub-array using the main value `$`.

Examples

- `@global.*`

  The global variable 'global' contains a special compound that contains all known global objects. This expressions retrieves a list of refs to all of them.

  Using `$global.*` would return values for the primitive ones instead of refs, and is not that useful.

- `$global.world.units.all[0].id`

  This expression is syntactically parsed into the following sequence:

```
tmp = global
tmp = @tmp.world  ; the world global ref
tmp = @tmp.units  ; the units field ref
tmp = @tmp.all    ; the all vector ref
tmp = @tmp[0]     ; the first unit object pointer ref
$tmp.id
```

  The only non-trivial step here is the last one. The last value of tmp is a reference to a pointer, and as described above, it delegates anything it does not directly understand to its target, adding an implicit step at runtime:

```
unit = @tmp._target
$unit.id
```

  A unit object does not define `$unit.id` directly either, so the final step falls back to:

```
idref = @unit.id
($ idref t)
```

  which retrieves a reference to the `id` field, and then evaluates its natural value.

  The result is that the expression returns the id value of the first unit in the vector as would be naturally expected.

  Using `@global.world.units.all[0].id` would have used `@tmp.id` as the last step, which would have skipped the `($ idref t)` call and returned a reference to the field.

- A simple `index-refers-to` example:

```
<stl-vector name='created_weapons' type-name='int32_t'
            index-refers-to='$global.world.raws.itemdefs.weapons[$]'/>
```

  This is used to define a vector with counts of created weapons.

  When it is displayed in the GUI, the tool evaluates the `index-refers-to` expression for every vector element, giving it the *element index* as `$`, and a reference to the vector itself as `$$` (here unused).

The expression straightforwardly uses that index to access another global vector and return one of its elements. It is then used by the GUI to add additional information to the info column.

- An example of `refers-to` and `_parent`:

```
<compound name='burrows'>
    <stl-vector name='list' pointer-type='burrow'/>
    <int32_t name='sel_index' refers-to='$$._parent.list[$]'/>
</compound>
```

This fragment of XML defines a compound with two fields, a vector and an int, which has a `refers-to` attribute. When that field is displayed in the GUI, it evaluates the expression in the attribute, giving it the *integer value* as `$`, and a *reference* to the integer field as `$$`.

The expression parses as:

```
tmp = $$            ; reference to the int32_t field
tmp = @tmp._parent
tmp = @tmp.list
$tmp[$]
```

Since the only way the GUI could get a reference to the field was to evaluate `@ref-to-burrows.sel_index`, that previous reference is stored in its "back" list, and `@tmp._parent` retrieves it. After that everything is simple.

- An example of `ref-target` with `aux-value`:

```
<int32_t name='race' ref-target='creature_raw'/>
<int16_t name='caste' ref-target='caste_raw' aux-value='$$.race'/>
```

The `race` field just specifies a type as `ref-target`, so the reference simply evaluates the `find-instance` helper of the `creature_raw`, passing it the race value as `$`.

In order to find the caste however, you need to first find a creature, which requires a race value. This value is supplied via the `aux-value` attribute into the `$$` argument to `find-instance`.

Since the value of the `caste` field will be passed through to the helper anyway, when evaluating `aux-value` the `$` argument is set to a *reference* to the holding field, and `$$` is set to its `_parent`. This means that `$$.race` in the context of `aux-value` is equivalent to `$$._parent.race` in the context of `refers-to`.

- A complex example of cross-references between arrays:

```
<struct-type type-name='caste_raw'>
    <compound name='body_info'>
        <stl-vector name='body_parts' pointer-type='body_part_raw'/>
    </compound>
    <compound name='bp_appearance'>
        <stl-vector name='modifiers' pointer-type='bp_appearance_modifier'/>

        <stl-vector name='modifier_idx' type-name='int32_t'
                    refers-to='$$._parent._parent.modifiers[$]'
                    index-refers-to='$$._parent.part_idx[$].refers-to'/>
        <stl-vector name='part_idx' type-name='int16_t'
                    refers-to='$$._global.body_info.body_parts[$]'/>
        <stl-vector name='layer_idx' type-name='int16_t'
                    refers-to='$$._parent._parent.part_idx[$$._key].refers-to.layers[$]'
                    index-refers-to='$$._parent.part_idx[$].refers-to'/>
    </compound>
</struct-type>
```

In order to understand this example it is first necessary to understand that `refers-to` specified on a vector is actually transplanted onto the implicitly constructed element tag:

```
<stl-vector name='part_idx'>
    <int16_t refers-to='$$._global.body_info.body_parts[$]'/>
</stl-vector>
```

Therefore, `$$` is a reference to the `<int16_t>` field, `$$._parent` is a reference to the vector, `$$._parent._parent` is a reference to the `bp_appearance` compound, etc.

The `$$._global...` works as an abbreviation that applies `_parent` until it reaches a globally defined type, which in this case is the current instance of the `caste_raw` struct.

NOTE: `$$._global._global` is the same as `$$._global`, i.e. repeated `_global` is a no-op. The latest version supports `_upglobal`, which is equivalent to `_parent._global`.

Thus, the `refers-to` link on the `part_idx` vector evaluates to the element of the `body_parts` vector, indexed by the *value* of the current `part_idx` vector item.

Likewise, the `refers-to` link on the `modifier_idx` vector goes back to the `bp_appearance` compound, and descends into the `modifiers` vector, using the value of the current item.

The `index-refers-to` link on the same `modifier_idx` vector highlights the shared indexing relation between the bottom vectors by linking to the part_idx vector via the current item *index*. Since this attribute is hosted by the vector itself, `$$` points at the vector, and only one `_parent` is needed to reach `bp_appearance`.

This link also demonstrates how the defined relations can be reused in other expressions by accessing the target of the `refers-to` link inside `part_idx`. When the `part_idx` vector is accessed simply as `$xxx.part_idx[foo]`, it evaluates as:

```
tmp = @xxx.part_idx
tmp = @tmp[foo]
($ tmp t)
```

thus returning just an integer value. However, if an additional dereference step is added, it turns to:

```
tmp = @xxx.part_idx
tmp = @tmp[foo]
obj = @tmp.refers-to
($ obj t)
```

which follows the `refers-to` link and evaluates its target.

Finally, the `layer_idx` vector, in addition to specifying the same `index-refers-to` link as `modifier_idx`, uses the link in `part_idx` to access other objects at its end:

```
refers-to='$$._parent._parent.part_idx[$$._key].refers-to.layers[$]'
```

Note how this link has to use two `_parent` steps again due to being attached to the element of the vector instead of the vector itself. It also has to use the `_key` attribute of the vector element to retrieve the current index in the vector, because here `$` holds the element value.

4.5 Updating DF-structures for a new DF version

Contents

4.5.1 General Process

Download the new versions. The scripts expect the following directory layout:

```
~userhome/
    Games/
        DF/
            df_linux/    - Current DF for linux
            df_windows/  - Current DF for windows
            df_osx/      - Current DF for osx

            metasm/      - Checkout of the library for ruby scripts
            df_misc/     - Checkout of the ruby scripts
            dfhack/      - DFHack checkout
```

- Use "new-release.pl v0.??.??" to automatically perform a number of steps. If you get a mismatch error from match-ctors.pl, see "STAGE 1".
- Start the linux DF version, and launch the tool.
- Execute (reset-state-annotation)
- Commit.
- Use the tool to verify that the layout of the compound globals on linux is correct, and update xml as necessary. Check that unit seems reasonable, etc. Compare the changes in g_src between releases. Delete redundant entries for some linux & osx globals in symbols.xml.
- Compile DFHack without plugins for windows and run devel/find-offsets to find globals there.
- With the windows version in wine, run (check-struct-sizes) to see if any objects changed their size. If nothing is obviously wrong, use (check-struct-sizes :annotate? t) to mark correctly sized objects ALIGNED.
- Check the rest of the document for info on finding still missing globals.
- Commit.
- Run make-csv.sh to update CSV files and verify vtable size and argument counts.
- Commit.

4.5.2 Running Dwarf Fortress

The lisp tool expects that the game is started in a mode where all allocated memory is automatically filled with a certain byte value.

On linux this is achieved by simply starting the game like this:

```
MALLOC_PERTURB_=45 ./df
```

Windows requires applying a patch to a copy of the executable like this:

```
cp -f 'Dwarf Fortress.exe' 'Dwarf_Fortress_malloc.exe'
ruby -I ~/Games/DF/metasm ~/Games/DF/df_patchmalloc.rb 'Dwarf_Fortress_malloc.exe'
```

4.5.3 Available Scripts

new-release.pl

Takes the full v0.??.?? version number as one required parameter.

Uses md5sum for linux and winedump for windows to find out the new hash and PE timestamp.

Use the stamps to create new sections in symbols.xml Also paste them into make-csv inside start.lisp

Creates an empty v0.??.??.lst, and change thes open-annotations filename in start.lisp.

Wipes linux/df.globals.xml empty of all the global definitions.

Runs make-scans.sh to find out addresses of vtables and many linux/osx globals, and pastes them into symbols.xml

make-scans.sh

Runs ruby and perl scripts to extract data from the executables, and writes the output to txt files in subdirectories.

make-csv.sh

Uses the lisp tool and some scripts to produce csv files with offsets for linux and windows. These are useful for manual lookup and some scripts.

start.sh

Starts the lisp tool. You may pass the process ID as a parameter to avoid the prompt.

make-keybindings.pl

Used by make-scans to extract the keybinding enum from g_src in form of df.keybindings.xml

match-ctors.pl

Used by make-scans to compare the extracted addresses of the compound linux/osx globals with a saved copy from a previous version and thus determine their names.

match-vtables.pl

Used by make-csv.sh to produce a file listing the addresses of all virtual methods in a compact form. Relies on csv files and data from make-scans.sh

4.5.4 STAGE 1. Linux compound globals

(done by new-release.pl normally)

Linux and OSX initalize and destruct their complex globals in a way that allows to determine their addresses by disassembling a small section of the executable. This is currently done by ruby scripts called from new-release.pl; it is also possible to do that via the lisp tool for linux.

The ruby scripts produce a raw dump of the global addresses as linux/ctors.txt. A perl script is then used to compare it with linux/ctors-base.txt (which is manually edited and committed into the repository), and thus derive the names of the globals by their order. The resulting data is written back to linux/ctors.txt, linux/df.globals.xml and linux/cglobals.txt (which is inserted into symbols.xml).

If the size of a global changes too much, or a new one is added in the middle, this matching may fail. In this case it is necessary to manually match and add the new names to ctors.txt and commit it as ctors-base.txt. After that, run make-scans.sh to rerun the scripts, and paste linux/cglobals.txt into symbols.xml.

OSX behaves exactly the same as linux in this respect.

4.5.5 STAGE 2. Old way to find Linux compound globals

(now mostly obsolete, retained as fallback and for historical interest)

Globals gps, enabler, gview and init are in the export table for linking with libgraphics, so they are immediately available in (browse @global.*).

Run (list-globals/linux), paste the results in linux/df.globals.xml, and immediately compare it to the old version from source control. The order of the globals is quite stable, so if sizes look similar, they can be guessed immediately.

The .bss compound section should be done except for 'announcements'.

Run (browse-dataseg). The first three -30000 are cursor. Following group of 6 are selection_rect. After that, at 16-aligned addresses are control_mode and game_mode. Tab the game ui to the most common two-pane mode, scroll to the end and find 0x30200. Within this dword ui_menu_width is byte 1, ui_area_map_width is byte 2.

(reload), (browse @global.*), look at the most important globals for misalignment. If found, fix it and delete old tables from symbols.xml.

4.5.6 STAGE 3. Linux primitive globals

Unpause the game for a moment to let various structures be initialized.

The fields can be found either by a straight memory search, or by looking in the area they are expected to be.

[A] The 'cur_year' area.

Located just before ui_building_assign_type.

1. cur_year / cur_year_tick

   (find-changes); step with dot; Enter; step; +; step; +; step; +; done

   look at values in bss, there will be cur_year_tick, and cur_year is 32 bytes before that.

2. process_jobs

   Designate a building for construction. Look after process_dig for an enabled boolean.

3. process_dig

   Step the game one step. Designate a tile for digging. Look after cur_year and before process_jobs.

   Note: this order because designating sometimes sets process_jobs too.

4. job_next_id / ui_workshop_job_cursor

   Find a workshop without jobs; (find-changes); add job; Enter; add job; +; add job; +; done Finds job_next_id and ui_workshop_job_cursor, the distinction is obvious.

   The ui_workshop_job_cursor is expected to be after cur_year_tick.

5. ui_workshop_in_add, ui_building_in_resize, ui_building_in_assign

   Expected to be in the area after ui_workshop_job_cursor, in this order. Change the relevant state in game and F5.

6. ui_building_item_cursor

   Find a cluttered workshop, t; (find-changes); move cursor down; Enter; cursor down; +; cursor down; +; done

   Expected to be right after ui_workshop_job_cursor.

7. current_weather

   Subtract 0x1c from cur_year address. Obviously, a big hack.

   It is best to use a save where the contents are non-zero and known to you.

[B] The ui_look_cursor area.

Located in the area of the 124 byte global before ui.

1. ui_look_cursor

   Like ui_building_item_cursor, but with a cluttered tile and k.

2. ui_selected_unit

   Find a place with many nearby units; (find-changes); v; Enter; v; new; ...; when returned to origin, 0; 1; 2...; done

   Expected to be before ui_look_cursor.

3. ui_unit_view_mode

   Select unit, page Gen; (find-changes); Inv; Enter; Prf; +; Wnd; +; done

   Expected to be after ui_selected_unit.

4. pause_state

   (find-changes); toggle pause; Enter; toggle; 0; toggle; 1; etc; done

   Expected to be in the area after ui_look_cursor.

[C] The window_x/y/z area.

Located right after ui_build_selector.

1. window_x, window_y, window_z

   Use k, move window view to upper left corner, then the cursor to bottom right as far as it can go without moving the view.

(find-changes); Shift-RightDown; Enter; Shift-RightDown; + 10; Shift-RightDown; + 10; done

Finds cursor and two variables in bss. Z is just after them.

[D] Random positions.

1. announcements

   Immediately follows d_init; starts 25 25 31 31 24 ...

4.5.7 STAGE 4. Primary windows compound globals

After aligning globals on linux, run (make-csv) to produce offset tables.

1. world

Set a nickname, search for it; the unit will have it at offset 0x1C. Then trace back to the unit vector, and subtract its offset.

2. ui

Open the 's'quad sidebar page. Navigate to a squad in world.squads.all, then backtrace and subtract the offset of ui.squads.list.

3. ui_build_selector

Start creating a building, up to the point of material selection. Find the material item through world and backtrack references until .bss.

4. ui_sidebar_menus

Select a unit in 'v', open inventory page, backtrack from unit_inventory_item, subtract offset of unit.inv_items.

5. ui_look_list

Put a 'k' cursor over a unit, backtrack to a 0x10 bytes object with pointer at offset 0xC, then to the global vector.

6. ui_advmode

In adventure mode, open the 'c'ompanions menu, then backtrack from world.units.active[0] (i.e. the player) via ui_advmode.companions.unit

Alternatively, look before ui_look_list for "0, 15" coming from the string.

7. enabler

(find-changes), resize the window, enter; resize width by +1 char, +; repeat until few candidates left; then done, select the renderer heap object and backtrack to enabler.renderer.

Alternatively, look before ui for clocks changing every frame.

8. map_renderer

Put a 'v' cursor exactly above a unit; backtrack from the unit object.

Alternatively, look before ui_advmode for the unit pointer list.

9. texture

Load the game with [GRAPHICS:YES] in init.txt, and example set. Then search for string "example/dwarves.bmp" and backtrack.

Alternatively, look between ui_build_selector and init.

4.5.8 STAGE 5. Secondary windows compound globals

These are too difficult to find by backtracking or search, so try looking in the expected area first:

1. timed_events

Look for a pointer vector around -0x54 before ui.

2. ui_building_assign_*

2a. ui_building_assign_is_marked

> Assign to zone, (find-changes), toggle 1st unit, enter; toggle 1st, 0; toggle 1st, 1; toggle 2nd, new; done
>
> The vector is expected to be just before ui.

2b. ui_building_assign_items

> Expected to be immediately before ui_building_assign_is_marked.

2c. ui_building_assign_units

> Start assigning units to a pasture, backtrack from one of the units.
>
> The vector is expected to be immediately before world.

2d. ui_building_assign_type

> The vector is expected to be 2nd vector immediately after ui_look_list.

3. gview

Immediately follows ui.

4. Init files

4a. d_init

> Follows world after a small gap (starts with flagarray).

4b. init

> Follows ui_build_selector after a small gap.

5. gps

Look at around offset ui_area_map_width+0x470 for pointers.

6. created_item_*

6a. created_item_type

Expected to be at around -0x40 before world.

6b. created_item_subtype

The first vector immediately after ui_look_list.

6c. created_item_mattype

Immediately before ui_sidebar_menus.

6d. created_item_matindex

Before ui, after timed_events.

6e. created_item_count

Immediately before timed_events.

4.5.9 STAGE 6. Windows primitive globals

Like linux primitives, except the ordering is completely different.

This section only describes the ordering heuristics; for memory search instructions see linux primitive globals.

[A] formation_next_id

Followed by ui_building_item_cursor, cur_year.

[B] interaction_instance_next_id...hist_figure_next_id

Contains window_x, ui_workshop_in_add.

[C] machine_next_id

Followed by ui_look_cursor, window_y.

[D] crime_next_id

Followed by, in this order (but with some gaps):

- ui_workshop_job_cursor
- current_weather (immediately after ui_workshop_job_cursor)
- process_dig
- process_jobs
- ui_building_in_resize
- ui_building_in_assign
- pause_state

[E] Random positions.

1. cur_year_tick

Look immediately before artifact_next_id.

2. window_z

   Look before proj_next_id.

3. ui_selected_unit

   Look just after squad_next_id.

4. ui_unit_view_mode

   Look just before hist_event_collection_next_id.

5. announcements

   Immediately follows d_init; starts 25 25 31 31 24 ...

4.6 Patching the DF binary

Writing scripts and plugins for DFHack is not the only way to modify Dwarf Fortress. Before DFHack, it was common for tools to manually patch the binary to change behaviour, and DFHack still contains tools to do this via the *binpatch* command.

Warning: We recommend using a script or plugin instead of a raw patch if at all possible - that way your work will work for many versions across multiple operating systems. There's a reason nobody has written patches since `0.34.11`!

Contents

4.6.1 Getting a patch

There are no binary patches available for Dwarf Fortress versions after 0.34.11

This system is kept for the chance that someone will find it useful, so some hints on how to write your own follow. This will require disassembly and decent skill in *memory research*.

- The patches are expected to be encoded in text format used by IDA.
- See Commit 8a9e3d1a728 for examples.
- Issue 546 is about the future of the binpatches, and may be useful reading.

If you want to write a patch, the armory patches discussed here and documented below would probably be the best place to start.

4.6.2 Using a patch

There are two methods to apply a patch.

Patching at runtime

The *binpatch* script checks, applies or removes binary patches directly in memory at runtime:

```
binpatch [check|apply|remove] <patchname>
```

If the name of the patch has no extension or directory separators, the script uses `hack/patches/<df-version>/<name>.dif`, thus auto-selecting the version appropriate for the currently loaded executable.

This is the preferred method; it's easier to debug, does not cause persistent problems, and leaves file checksums alone. As with many other commands, users can simply add it to *dfhack*.init* to reapply the patch every time DF is run.

Patching on disk

Warning: This method of patching is deprecated, and may be removed without notice. You should use the runtime patching option above.

DFHack includes a small stand-alone utility for applying and removing binary patches from the game executable. Use it from the regular operating system console:

`binpatch check "Dwarf Fortress.exe" patch.dif` Checks and prints if the patch is currently applied.

`binpatch apply "Dwarf Fortress.exe" patch.dif` Applies the patch, unless it is already applied or in conflict.

`binpatch remove "Dwarf Fortress.exe" patch.dif` Removes the patch, unless it is already removed.

If you use a permanent patch under OSX or Linux, you must update `symbols.xml` with the new checksum of the executable. Find the relevant section, and add a new line:

```
<md5-hash value='????????????????????????????????'/>
```

In order to find the correct value of the hash, look into stderr.log; DFHack prints an error there if it does not recognize the hash.

4.6.3 Tools reliant on binpatches

Some DFHack tools require the game to be patched to work. As no patches are currently available, the full description of each is included here.

fix-armory

Enables a fix for storage of squad equipment in barracks.

Specifically, it prevents your haulers from moving squad equipment to stockpiles, and instead queues jobs to store it on weapon racks, armor stands, and in containers.

Note: In order to actually be used, weapon racks have to be patched and manually assigned to a squad. See *gui/assign-rack*.

Note that the buildings in the armory are used as follows:

- Weapon racks (when patched) are used to store any assigned weapons. Each rack belongs to a specific squad, and can store up to 5 weapons.
- Armor stands belong to specific squad members and are used for armor and shields.
- Cabinets are used to store assigned clothing for a specific squad member. They are **never** used to store owned clothing.
- Chests (boxes, etc) are used for a flask, backpack or quiver assigned to the squad member. Due to a probable bug, food is dropped out of the backpack when it is stored.

Warning: Although armor stands, cabinets and chests properly belong only to one squad member, the owner of the building used to create the barracks will randomly use any containers inside the room. Thus, it is recommended to always create the armory from a weapon rack.

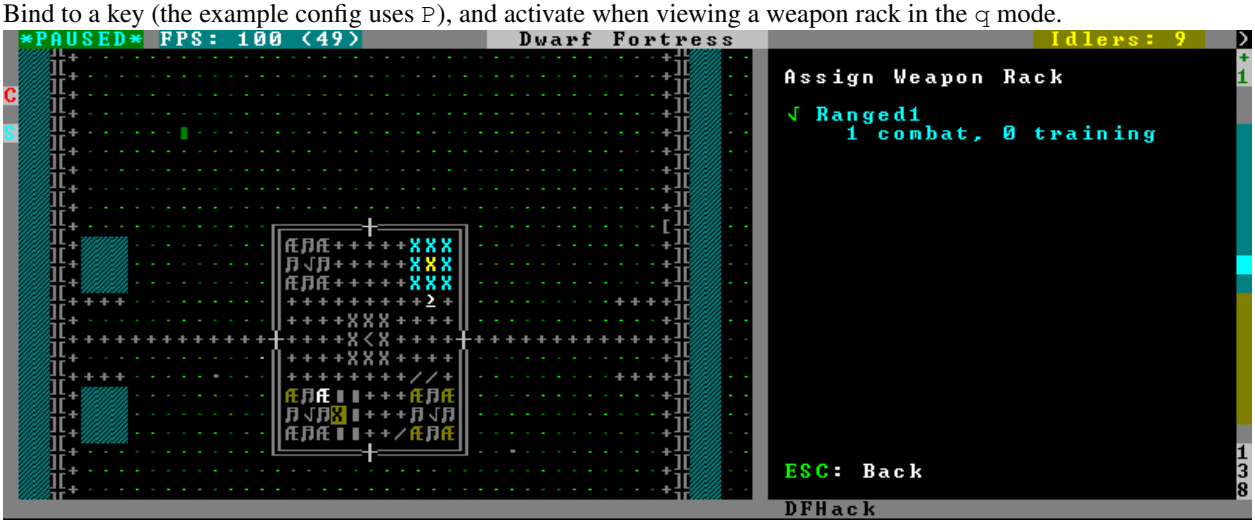
Contrary to the common misconception, all these uses are controlled by the *Individual Equipment* usage flag. The *Squad Equipment* flag is actually intended for ammo, but the game does even less in that area than for armor and weapons. This plugin implements the following rules almost from scratch:

- Combat ammo is stored in chests inside rooms with Squad Equipment enabled.
- If a chest is assigned to a squad member due to Individual Equipment also being set, it is only used for that squad's ammo; otherwise, any squads with Squad Equipment on the room will use all of the chests at random.
- Training ammo is stored in chests inside archery ranges designated from archery targets, and controlled by the same Train flag as archery training itself. This is inspired by some defunct code for weapon racks.

There are some minor traces in the game code to suggest that the first of these rules is intended by Toady; the rest are invented by this plugin.

gui/assign-rack

Bind to a key (the example config uses `P`), and activate when viewing a weapon rack in the `q` mode.



This script is part of a group of related fixes to make the armory storage work again. The existing issues are:

- Weapon racks have to each be assigned to a specific squad, like with beds/boxes/armor stands and individual squad members, but nothing in the game does this. This issue is what this script addresses.
- Even if assigned by the script, **the game will unassign the racks again without a binary patch**. This patch is called `weaponrack-unassign`, and has not been updated since 0.34.11. See Bug 1445 for more info.
- Haulers still take equipment stored in the armory away to the stockpiles, unless *fix-armory* is used.

The script interface simply lets you designate one of the squads that are assigned to the barracks/armory containing the selected stand as the intended user. In order to aid in the choice, it shows the number of currently assigned racks for every valid squad.