DFHack Documentation

Release 0.47.05-r5

The DFHack Team

2022-05-05

Contents

DFHack is a memory editing library for Dwarf Fortress that provides a unified, cross-platform environment where tools can be developed to extend the game. The default distribution contains a variety of tools, including bugfixes, interface improvements, automation tools, modding tools, and more. There are also a variety of third-party tools available.

Part I

Quick Links

- Downloads
- *Installation guide*
- Source code (**important:** read *Compiling DFHack* before attempting to build from source)
- Bay 12 forums thread
- Bug tracker

Part II

User Manual

CHAPTER 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 on GitHub, and can be downloaded from the releases page - see *Installing DFHack* for installation instructions. This is also where the DFHack bug tracker is hosted.

All new releases are announced in the Bay12 forums thread, which is also a good place for discussion and questions.

For users, DFHack 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 open-source licences*, contributions are welcome.

Contents

1.1 Getting started

See *Installing DFHack* for details on installing DFHack.

Once DFHack is installed, it extends DF with a console that can be used to run commands. On Windows, this console will open automatically when DF is started. On Linux and macOS, you will need to run the `dfhack` script from a terminal (instead of the `df` script included with DF), and that terminal will be used by the DFHack 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.

1.2 Getting help

There are several support channels available - see *Getting Support* for details.

CHAPTER 2

Installing DFHack

2.1 Requirements

DFHack supports Windows, Linux, and macOS, and both 64-bit and 32-bit builds of Dwarf Fortress.

DFHack releases generally only support the version of Dwarf Fortress that they are named after. For example, DFHack 0.40.24-r5 only supported DF 0.40.24. DFHack releases *never* support newer versions of DF, because DFHack requires data about DF that is only possible to obtain after DF has been released. Occasionally, DFHack releases will be able to maintain support for older versions of DF - for example, DFHack 0.34.11-r5 supported both DF 0.34.11 and 0.34.10. For maximum stability, you should usually use the latest versions of both DF and DFHack.

2.1.1 Windows

- DFHack only supports the SDL version of Dwarf Fortress. The "legacy" version will *not* work with DFHack (the "small" SDL version is acceptable, however).
- Windows XP and older are *not* supported, due in part to a Visual C++ 2015 bug

The Windows build of DFHack should work under Wine on other operating systems, although this is not tested very often. It is recommended to use the native build for your operating system instead.

2.1.2 Linux

Generally, DFHack should work on any modern Linux distribution. There are multiple release binaries provided - as of DFHack 0.47.04-r1, there are built with GCC 7 and GCC 4.8 (as indicated by the `gcc` component of their filenames). Using the newest build that works on your system is recommended. The GCC 4.8 build is built on Ubuntu 14.04 and targets an older glibc, so it should work on older distributions.

In the event that none of the provided binaries work on your distribution, you may need to *compile DFHack from source*.

2.1.3 macOS

OS X 10.6.8 or later is required.

2.2 Downloading DFHack

Stable builds of DFHack are available on GitHub. GitHub has been known to change their layout periodically, but as of July 2020, downloads are available at the bottom of the release notes for each release, under a section named "Assets" (which you may have to expand). The name of the file indicates which DF version, platform, and architecture the build supports - the platform and architecture (64-bit or 32-bit) **must** match your build of DF. The DF version should also match your DF version - see *above* for details. For example:

- `dfhack-0.47.04-r1-Windows-64bit.zip` supports 64-bit DF on Windows
- `dfhack-0.47.04-r1-Linux-32bit-gcc-7.tar.bz2` supports 32-bit DF on Linux (see *Linux* for details on the GCC version indicator)

The DFHack website also provides links to unstable builds. These files have a different naming scheme, but the same restrictions apply (e.g. a file named `Windows64` is for 64-bit Windows DF).

Warning: Do *not* download the source code from GitHub, either from the releases page or by clicking "Download ZIP" on the repo homepage. This will give you an incomplete copy of the DFHack source code, which will not work as-is. (If you want to compile DFHack instead of using a pre-built release, see *Compiling DFHack* for instructions.)

2.3 Installing DFHack

When you *download DFHack*, you will end up with a release archive (a `.zip` file on Windows, or a `.tar.bz2` file on other platforms). Your operating system should have built-in utilities capable of extracting files from these archives.

The release archives contain several files and folders, including a `hack` folder, a `dfhack-config` folder, and a `dfhack.init-example` file. To install DFHack, copy all of the files from the DFHack archive into the root DF folder, which should already include a `data` folder and a `raw` folder, among other things. Some packs and other redistributions of Dwarf Fortress may place DF in another folder, so ensure that the `hack` folder ends up next to the `data` folder.

Note: On Windows, installing DFHack will overwrite `SDL.dll`. This is intentional and necessary for DFHack to work, so be sure to choose to overwrite `SDL.dll` if prompted. (If you are not prompted, you may be installing DFHack in the wrong place.)

2.4 Uninstalling DFHack

Uninstalling DFHack essentially involves reversing what you did to install DFHack. On Windows, replace `SDL.dll` with `SDLreal.dll` first. Then, you can remove any files that were part of the DFHack archive. DFHack does not currently maintain a list of these files, so if you want to completely remove them, you should consult the DFHack archive that you installed for a full list. Generally, any files left behind should not negatively affect DF.

2.5 Upgrading DFHack

The recommended approach to upgrade DFHack is to uninstall DFHack first, then install the new version. This will ensure that any files that are only part of the older DFHack installation do not affect the new DFHack installation (although this is unlikely to occur).

It is also possible to overwrite an existing DFHack installation in-place. To do this, follow the installation instructions above, but overwrite all files that exist in the new DFHack archive (on Windows, this includes `SDL.dll` again).

Note: You may wish to make a backup of your `dfhack-config` folder first if you have made changes to it. Some archive managers (e.g. Archive Utility on macOS) will overwrite the entire folder, removing any files that you have added.

2.6 Pre-packaged DFHack installations

There are several packs available that include DF, DFHack, and other utilities. If you are new to Dwarf Fortress and DFHack, these may be easier to set up. Note that these packs are not maintained by the DFHack team and vary in their release schedules and contents. Some may make significant configuration changes, and some may not include DFHack at all.

2.7 Linux packages

Third-party DFHack packages are available for some Linux distributions, including in:

- AUR, for Arch and related distributions
- RPM Fusion, for Fedora and related distributions

Note that these may lag behind DFHack releases. If you want to use a newer version of DFHack, we generally recommended installing it in a clean copy of DF in your home folder. Attempting to upgrade an installation of DFHack from a package manager may break it.

CHAPTER 3

Getting Support

DFHack has several ways to get help online, including:

- The DFHack Discord server
- The `#dfhack` IRC channel on Libera
- **GitHub:**
  - for bugs, use the issue tracker
  - for more open-ended questions, use the discussion board. Note that this is a relatively-new feature as of 2021, but maintainers should still be notified of any discussions here.
- The DFHack thread on the Bay 12 Forum

Some additional, but less DFHack-specific, places where questions may be answered include:

- The /r/dwarffortress questions thread on Reddit
- If you are using a starter pack, the relevant thread on the Bay 12 Forum - see the DF Wiki for a list of these threads

When reaching out to any support channels regarding problems with DFHack, please remember to provide enough details for others to identify the issue. For instance, specific error messages (copied text or screenshots) are helpful, as well as any steps you can follow to reproduce the problem. Sometimes, log output from `stderr.log` in the DF folder can point to the cause of issues as well.

Some common questions may also be answered in documentation, including:

- This documentation (online here; search functionality available here)
- The DF wiki

CHAPTER 4

DFHack Core

Contents

4.1 Command Implementation

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 because 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.

4.2 Using DFHack Commands

DFHack commands can be executed in a number of ways:

1. Typing the command into the DFHack console (see below)
2. From the OS terminal (see below)
3. Pressing a key combination set up with *keybinding*
4. From one of several *Init Files*, automatically
5. Using *script* to run a batch of commands from a file

4.2.1 The DFHack Console

The command line has some nice line editing capabilities, including history that's preserved between different runs of DF - use ↑ and ↓ to go through the history.

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. The non-whitespace characters following the `:` are the command name, and the remaining part of the line is used verbatim as the first argument. This is very useful for the *lua* and *ruby* commands. As an example, the following two command lines are exactly equivalent:

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

4.2.2 Using an OS terminal

DFHack commands can be run from an OS terminal at startup, using '+ args', or at any other time using the `dfhack-run` executable.

If DF/DFHack is started with arguments beginning with +, the remaining text is treated as a command in the DFHack console. It is possible to use multiple such commands, which are split on +. For example:

```
./dfhack +load-save region1
"Dwarf Fortress.exe" +devel/print-args Hello! +enable workflow
```

The first example (*nix), *load-save*, skips the main menu and loads `region1` immediately. The second (Windows) example prints *Hello!* in the DFHack console, and *enables workflow*. Note that the `:foo` syntax for whitespace in arguments is not compatible with '+ args'.

dfhack-run

If DF and DFHack are already running, calling `dfhack-run my command` in an external terminal is equivalent to calling `my command` in the DFHack console. Direct use of the DFHack console is generally easier, but `dfhack-run` can be useful in a variety of circumstances:

- if the console is unavailable
  - with the init setting `PRINT_MODE:TEXT`
  - while running an interactive command (e.g. *liquids* or *tiletypes*)
- from external programs or scripts
- if DF or DFHack are not responding

Examples:

```
./dfhack-run cursecheck
dfhack-run kill-lua
```

The first (*nix) example *checks for vampires*; the second (Windows) example uses *kill-lua* to stop a Lua script.

Note: `dfhack-run` attempts to connect to a server on TCP port 5000. If DFHack was unable to start this server, `dfhack-run` will not be able to connect. This could happen if you have other software listening on port 5000, or if you have multiple copies of DF running simultaneously. To assign a different port, see *Server configuration*.

4.3 Built-in Commands

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

4.3.1 alias

The `alias` command allows configuring aliases to other DFHack commands. Aliases are resolved immediately after built-in commands, which means that an alias cannot override a built-in command, but can override a command implemented by a plugin or script.

Usage:

`alias list` lists all configured aliases

`alias add <name> <command> [arguments...]` adds an alias

`alias replace <name> <command> [arguments...]` replaces an existing alias with a new command, or adds the alias if it does not already exist

`alias delete <name>` removes the specified alias

Aliases can be given additional arguments when created and invoked, which will be passed to the underlying command in order. An example with *devel/print-args*:

```
[DFHack]# alias add pargs devel/print-args example
[DFHack]# pargs text
example
text
```

4.3.2 cls

Clear the terminal. Does not delete command history.

4.3.3 die

Instantly kills DF without saving.

4.3.4 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. Passing plugin names to these commands will enable or disable the specified plugins. For example, to enable the *manipulator* plugin:

```
enable manipulator
```

It is also possible to enable or disable multiple plugins at once:

```
enable manipulator search
```

4.3.5 fpause

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

4.3.6 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`.

4.3.7 hide

Hides the DFHack terminal window. Only available on Windows.

4.3.8 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.

4.3.9 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.

4.3.10 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.

Note that plugins do not maintain their enabled state if they are reloaded, so you may need to use *enable* to re-enable a plugin after reloading it.

4.3.11 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.

4.3.12 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.

4.3.13 sc-script

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

4.3.14 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`

4.3.15 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*.

4.3.16 type

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

4.3.17 Other Commands

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

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

4.4 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.

4.4.1 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.

4.4.2 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 alphabetical 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*.

4.4.3 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.

4.4.4 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 (e.g. *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.

4.5 Configuration Files

Some DFHack settings can be changed by modifying files in the `dfhack-config` folder (which is in the DF folder). The default versions of these files, if they exist, are in `dfhack-config/default` and are installed when DFHack starts if necessary.

4.5.1 Script paths

Script paths are folders that DFHack searches to find a script when a command is run. By default, the following folders are searched, in order (relative to the root DF folder):

1. `data/save/<region folder>/raw/scripts` (only if a save is loaded)
2. `raw/scripts`
3. `hack/scripts`

For example, if `teleport` is run, these folders are searched in order for `teleport.lua` or `teleport.rb`, and the first matching file is run.

Script paths can be added by modifying `dfhack-config/script-paths.txt`. Each line should start with one of these characters:

- `+`: adds a script path that is searched *before* the default paths (above)
- `-`: adds a script path that is searched *after* the default paths
- `#`: a comment (the line is ignored)

Paths can be absolute or relative - relative paths are interpreted relative to the root DF folder.

Tip

When developing scripts in the dfhack/scripts repo, it may be useful to add the path to your local copy of the repo with `+`. This will allow you to make changes in the repo and have them take effect immediately, without needing to re-install or copy scripts over manually.

Note that `script-paths.txt` is only read at startup, but the paths can also be modified programmatically at any time through the *Lua API*.

4.6 Environment Variables

DFHack's behavior can be adjusted with some environment variables. For example, on UNIX-like systems:

```
DFHACK_SOME_VAR=1 ./dfhack
```

- `DFHACK_PORT`: the port to use for the RPC server (used by `dfhack-run` and *remotefortressreader* among others) instead of the default `5000`. As with the default, if this port cannot be used, the server is not started. See *DFHack Remote Interface* for more details.
- `DFHACK_DISABLE_CONSOLE`: if set, the DFHack console is not set up. This is the default behavior if `PRINT_MODE:TEXT` is set in `data/init/init.txt`. Intended for situations where DFHack cannot run in a terminal window.
- `DFHACK_HEADLESS`: if set, and `PRINT_MODE:TEXT` is set, DF's display will be hidden, and the console will be started unless `DFHACK_DISABLE_CONSOLE` is also set. Intended for non-interactive gameplay only.
- `DFHACK_NO_GLOBALS`, `DFHACK_NO_VTABLES`: ignores all global or vtable addresses in `symbols.xml`, respectively. Intended for development use - e.g. to make sure tools do not crash when these addresses are missing.
- `DFHACK_NO_DEV_PLUGINS`: if set, any plugins from the plugins/devel folder that are built and installed will not be loaded on startup.
- `DFHACK_LOG_MEM_RANGES` (macOS only): if set, logs memory ranges to `stderr.log`. Note that *devel/lsmem* can also do this.
- `DFHACK_ENABLE_LUACOV`: if set, enables coverage analysis of Lua scripts. Use the *devel/luacov* script to generate coverage reports from the collected metrics.

Other (non-DFHack-specific) variables that affect DFHack:

- `TERM`: if this is set to `dumb` or `cons25` on *nix, the console will not support any escape sequences (arrow keys, etc.).
- `LANG`, `LC_CTYPE`: if either of these contain "UTF8" or "UTF-8" (not case sensitive), `DF2CONSOLE()` will produce UTF-8-encoded text. Note that this should be the case in most UTF-8-capable *nix terminal emulators already.

4.7 Miscellaneous Notes

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

- 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. *This feature will be removed in a future version.* (see Issue 731)
- The binaries for 0.40.15-r1 to 0.34.11-r4 are on DFFD. Older versions are available here. *These files will eventually be migrated to GitHub.* (see Issue 473)

CHAPTER 5

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

- *automaterial*
- *automelt*
- *autotrade*
- *buildingplan*
- *command-prompt*
- *confirm*
- *debug*
- *embark-assistant*
- *embark-tools*
- *follow*
- *hotkeys*
- *manipulator*
- *mousequery*
- *nopause*
- *rename*
- *rendermax*
- *resume*
- *ruby*
- *search*
- *sort-items*
- *sort-units*
- *stockpiles*
- *stocks*
- *title-folder*
- *title-version*
- *trackstop*

5.1 Bugfixes

- *fix-armory*
- *fix-unit-occupancy*
- *fixveins*
- *petcapRemover*
- *tweak*

5.1.1 fix-armory

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

5.1.2 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.

5.1.3 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.

5.1.4 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

5.1.5 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

burrow-name-cancel Implements the "back" option when renaming a burrow, which currently does nothing (Bug 1518)

cage-butcher Adds an option to butcher units when viewing cages with `q`

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

condition-material Fixes a crash in the work order contition material list (Bug 9905).

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.

do-job-now Adds a job priority toggle to the jobs list

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

hotkey-clear Adds an option to clear currently-bound hotkeys (in the `H` menu)

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

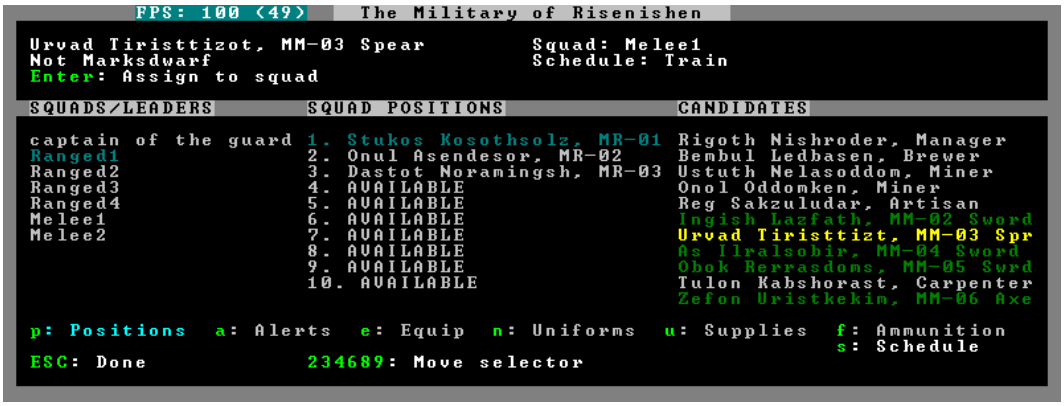
kitchen-prefs-all Adds an option to toggle cook/brew for all visible items in kitchen preferences

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)

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

partial-items Displays percentages on partially-consumed items such as hospital cloth

reaction-gloves Fixes reactions to produce gloves in sets with correct handedness (Bug 6273)

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, if the map view is near enough to its previous position.

stone-status-all Adds an option to toggle the economic status of all stones

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

5.2 Data inspection and visualizers

- *blueprint*
- *cursecheck*
- *flows*
- *isoworldremote*
- *probe*
- *prospect*
- *remotefortressreader*
- *reveal*
- *showmood*
- *spectate*

- *stonesense*

5.2.1 blueprint

The `blueprint` command exports the structure of a portion of your fortress in a blueprint file that you (or anyone else) can later play back with *quickfort*.

Blueprints are `.csv` or `.xlsx` files created in the `blueprints` subdirectory of your DF folder. The map area to turn into a blueprint is either selected interactively with the `blueprint gui` command or, if the GUI is not used, starts at the active cursor location and extends right and down for the requested width and height.

Usage:

```
blueprint <width> <height> [<depth>] [<name> [<phases>]] [<options>]
blueprint gui [<name> [<phases>]] [<options>]
```

Examples:

`blueprint gui` Runs *gui/blueprint*, the interactive frontend, where all configuration for a `blueprint` command can be set visually and interactively.

`blueprint 30 40 bedrooms` Generates blueprints for an area 30 tiles wide by 40 tiles tall, starting from the active cursor on the current z-level. Blueprints are written sequentially to `bedrooms.csv` in the `blueprints` directory.

`blueprint 30 40 bedrooms dig --cursor 108,100,150` Generates only the `#dig` blueprint in the `bedrooms.csv` file, and the start of the blueprint area is set to a specific value instead of using the in-game cursor position.

Positional Parameters:

`width` Width of the area (in tiles) to translate.

`height` Height of the area (in tiles) to translate.

`depth` Number of z-levels to translate. Positive numbers go *up* from the cursor and negative numbers go *down*. Defaults to 1 if not specified, indicating that the blueprint should only include the current z-level.

`name` Base name for blueprint files created in the `blueprints` directory. If no name is specified, "blueprint" is used by default. The string must contain some characters other than numbers so the name won't be confused with the optional `depth` parameter.

Phases:

If you want to generate blueprints only for specific phases, add their names to the commandline, anywhere after the blueprint base name. You can list multiple phases; just separate them with a space.

`dig` Generate quickfort `#dig` blueprints for digging natural stone.

`carve` Generate quickfort `#dig` blueprints for smoothing and carving.

`build` Generate quickfort `#build` blueprints for constructions and buildings.

`place` Generate quickfort `#place` blueprints for placing stockpiles.

`zone` Generate quickfort `#zone` blueprints for designating zones.

`query` Generate quickfort `#query` blueprints for configuring rooms.

If no phases are specified, phases are autodetected. For example, a `#place` blueprint will be created only if there are stockpiles in the blueprint area.

Options:

`-c, --cursor <x>,<y>,<z>`: Use the specified map coordinates instead of the current cursor position for the upper left corner of the blueprint range. If this option is specified, then an active game map cursor is not necessary.

`-e, --engrave`: Record engravings in the `carve` phase. If this option is not specified, engravings are ignored.

`-f, --format <format>`: Select the output format of the generated files. See the `Output formats` section below for options. If not specified, the output format defaults to "minimal", which will produce a small, fast `.csv` file.

`-h, --help`: Show command help text.

`-s, --playback-start <x>,<y>,<comment>`: Specify the column and row offsets (relative to the upper-left corner of the blueprint, which is `1,1`) where the player should put the cursor when the blueprint is played back with *quickfort*, in *quickfort start marker* format, for example: `10,10,central stairs`. If there is a space in the comment, you will need to surround the parameter string in double quotes: `"-s10,10,central stairs"` or `--playback-start "10,10,central stairs"` or `"--playback-start=10,10,central stairs"`.

`-t, --splitby <strategy>`: Split blueprints into multiple files. See the `Splitting output into multiple files` section below for details. If not specified, defaults to "none", which will create a standard quickfort *multi-blueprint* file.

Output formats:

Here are the values that can be passed to the `--format` flag:

`minimal` Creates `.csv` files with minimal file size that are fast to read and write. This is the default.

`pretty` Makes the blueprints in the `.csv` files easier to read and edit with a text editor by adding extra spacing and alignment markers.

Splitting output into multiple files:

The `--splitby` flag can take any of the following values:

`none` Writes all blueprints into a single file. This is the standard format for quickfort fortress blueprint bundles and is the default.

`phase` Creates a separate file for each phase.

5.2.2 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).

ids Print the creature and race IDs.

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.

5.2.3 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.

5.2.4 isoworldremote

A plugin that implements a *remote API* used by Isoworld.

5.2.5 probe

This plugin provides multiple commands that print low-level properties of the selected objects.

- `probe`: prints some properties of the tile selected with `k`. Some of these properties can be passed into *tiletypes*.
- `cprobe`: prints some properties of the unit selected with `v`, as well as the IDs of any worn items. *gui/gm-unit* and *gui/gm-editor* are more complete in-game alternatives.
- `bprobe`: prints some properties of the building selected with `q` or `t`. *gui/gm-editor* is a more complete in-game alternative.

5.2.6 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 were 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.

5.2.7 remotefortressreader

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

5.2.8 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. Note that tiles behind constructed walls are also revealed as a workaround for Bug 1871.

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

5.2.9 showmood

Shows all items needed for the currently active strange mood.

5.2.10 spectate

Simple plugin to automate following random dwarves. Most of the time things will be weighted towards z-levels with the highest job activity. Simply enter the `spectate` command to toggle the plugin's state.

5.2.11 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.

5.3 Job and Fortress management

5.3.1 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
```

5.3.2 autochop

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

Open the dashboard by running:

```
enable autochop
```

The plugin must be activated (with `d-t-c-a`) 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 autochop` to your dfhack.init there will be a hotkey to open the dashboard from the chop designation menu.

5.3.3 autoclothing

Automatically manage clothing work orders, allowing the user to set how many of each clothing type every citizen should have. Usage:

```
autoclothing <material> <item> [number]
```

Examples:

- **`autoclothing cloth "short skirt" 10`:** Sets the desired number of cloth short skirts available per citizen to 10.
- **`autoclothing cloth dress`:** Displays the currently set number of cloth dresses chosen per citizen.

5.3.4 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.

Keybinding: CtrlShiftK

visible Only process items that are not hidden.

hidden Only process hidden items.

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

`autodump-destroy-item` destroys the selected item, which may be selected in the `k` list, or inside a container. If called again before the game is resumed, cancels destruction of the item.

Keybinding: CtrlK

5.3.5 autofarm

Automatically handles crop selection in farm plots based on current plant stocks, and selects crops for planting if current stock is below a threshold. Selected crops are dispatched on all farmplots. (Note that this plugin replaces an older Ruby script of the same name.)

Use the *enable* or *disable* commands to change whether this plugin is enabled.

Usage:

- **`autofarm runonce`:** Updates all farm plots once, without enabling the plugin
- **`autofarm status`:** Prints status information, including any applied limits
- **`autofarm default 30`:** Sets the default threshold

- **`autofarm threshold 150 helmet_plump tail_pig`:** Sets thresholds of individual plants

5.3.6 autogems

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

See *gui/autogems* for a configuration UI. If necessary, the `autogems-reload` command reloads the configuration file produced by that script.

5.3.7 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.

5.3.8 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 equipment (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.

5.3.9 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.

5.3.10 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.

5.3.11 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.

5.3.12 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`)

Keybinding: AltM -> `"dwarfmonitor prefs"` in `dwarfmode/Default`

Keybinding: CtrlF -> `"dwarfmonitor stats"` in `dwarfmode/Default`

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.

5.3.13 dwarfvet

Enables Animal Caretaker functionality

Always annoyed your dragons become useless after a minor injury? Well, with dwarfvet, your animals become first rate members of your fort. It can also be used to train medical skills.

Animals need to be treated in an animal hospital, which is simply a hospital that is also an animal training zone. The console will print out a list on game load, and whenever one is added or removed. Dwarfs must have the Animal Caretaker labor to treat animals. Normal medical skills are used (and no experience is given to the Animal Caretaker skill).

Options:

enable Enables Animal Caretakers to treat and manage animals

disable Turns off the plguin

report Reports all zones that the game considers animal hospitals

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).

5.3.14 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.

5.3.15 job-duplicate

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

Keybinding: CtrlD

5.3.16 job-material

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

Invoked as:

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

Keybinding: ShiftA -> "`job-material ALUNITE`"

Keybinding: ShiftM -> "`job-material MICROCLINE`"

Keybinding: ShiftD -> "`job-material DACITE`"

Keybinding: ShiftR -> "`job-material RHYOLITE`"

Keybinding: ShiftI -> "`job-material CINNABAR`"

Keybinding: ShiftB -> "`job-material COBALTITE`"

Keybinding: ShiftO -> "`job-material OBSIDIAN`"

Keybinding: ShiftT -> "`job-material ORTHOCLASE`"

Keybinding: ShiftG -> "`job-material GLASS_GREEN`"

- 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.

5.3.17 labormanager

Automatically manage dwarf labors to efficiently complete jobs. Labormanager is derived from autolabor (above) but uses a completely different approach to assigning jobs to dwarves. While autolabor tries to keep as many dwarves busy as possible, labormanager instead strives to get jobs done as quickly as possible.

Labormanager frequently scans the current job list, current list of dwarfs, and the map to determine how many dwarves need to be assigned to what labors in order to meet all current labor needs without starving any particular type of job.

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

Simple usage:

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

disable labormanager Disables the plugin.

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

The default priorities for each labor vary (some labors are higher priority by default than others). The way the plugin works is that, once it determines how many of each labor is needed, it then sorts them by adjusted priority. (Labors other than hauling have a bias added to them based on how long it's been since they were last used, to prevent job starvation.) The labor with the highest priority is selected, the "best fit" dwarf for that labor is assigned to that labor, and then its priority is *halved*. This process is repeated until either dwarfs or labors run out.

Because there is no easy way to detect how many haulers are actually needed at any moment, the plugin always ensures that at least one dwarf is assigned to each of the hauling labors, even if no hauling jobs are detected. At least one dwarf is always assigned to construction removing and cleaning because these jobs also cannot be easily detected. Lever pulling is always assigned to everyone. Any dwarfs for which there are no jobs will be assigned hauling, lever pulling, and cleaning labors. If you use animal trainers, note that labormanager will misbehave if you assign specific trainers to specific animals; results are only guaranteed if you use "any trainer", and animal trainers will probably be overallocated in any case.

Labormanager also sometimes assigns extra labors to currently busy dwarfs so that when they finish their current job, they will go off and do something useful instead of standing around waiting for a job.

There is special handling to ensure that at least one dwarf is assigned to haul food whenever food is detected left in a place where it will rot if not stored. This will cause a dwarf to go idle if you have no storepiles to haul food to.

Dwarfs who are unable to work (child, in the military, wounded, handless, asleep, in a meeting) are entirely excluded from labor assignment. Any dwarf explicitly assigned to a burrow will also be completely ignored by labormanager.

The fitness algorithm for assigning jobs to dwarfs generally attempts to favor dwarfs who are more skilled over those who are less skilled. It also tries to avoid assigning female dwarfs with children to jobs that are "outside", favors assigning "outside" jobs to dwarfs who are carrying a tool that could be used as a weapon, and tries to minimize how often dwarfs have to reequip.

Labormanager automatically determines medical needs and reserves health care providers as needed. Note that this may cause idling if you have injured dwarfs but no or inadequate hospital facilities.

Hunting is never assigned without a butchery, and fishing is never assigned without a fishery, and neither of these labors is assigned unless specifically enabled.

The method by which labormanager determines what labor is needed for a particular job is complicated and, in places, incomplete. In some situations, labormanager will detect that it cannot determine what labor is required. It will, by default, pause and print an error message on the dfhack console, followed by the message "LABORMANAGER: Game paused so you can investigate the above message.". If this happens, please open an issue on github, reporting the lines that immediately preceded this message. You can tell labormanager to ignore this error and carry on by typing `labormanager pause-on-error no`, but be warned that some job may go undone in this situation.

Advanced usage:

labormanager enable Turn plugin on.

labormanager disable Turn plugin off.

labormanager priority <labor> <value> Set the priority value (see above) for labor <labor> to <value>.

labormanager reset <labor> Reset the priority value of labor <labor> to its default.

labormanager reset-all Reset all priority values to their defaults.

labormanager allow-fishing Allow dwarfs to fish. *Warning* This tends to result in most of the fort going fishing.

labormanager forbid-fishing Forbid dwarfs from fishing. Default behavior.

labormanager allow-hunting Allow dwarfs to hunt. *Warning* This tends to result in as many dwarfs going hunting as you have crossbows.

labormanager forbid-hunting Forbid dwarfs from hunting. Default behavior.

labormanager list Show current priorities and current allocation stats.

labormanager pause-on-error yes Make labormanager pause if the labor inference engine fails. See above.

labormanager pause-on-error no Allow labormanager to continue past a labor inference engine failure.

5.3.18 nestboxes

Automatically scan for and forbid fertile eggs incubating in a nestbox. Toggle status with *enable* or *disable*.

5.3.19 orders

A plugin for manipulating manager orders.

Subcommands:

export NAME Exports the current list of manager orders to a file named `dfhack-config/orders/NAME.json`.

import NAME Imports manager orders from a file named `dfhack-config/orders/NAME.json`.

clear Deletes all manager orders in the current embark.

sort Sorts current manager orders by repeat frequency so daily orders don't prevent other orders from ever being completed: one-time orders first, then yearly, seasonally, monthly, then finally daily.

You can keep your orders automatically sorted by adding the following command to your `onMapLoad.init` file:

```
repeat -name orders-sort -time 1 -timeUnits days -command [ orders sort ]
```

5.3.20 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.

5.3.21 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.

Keybinding: CtrlC

5.3.22 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.

5.3.23 tailor

Whenever the bookkeeper updates stockpile records, this plugin will scan every unit in the fort, count up the number that are worn, and then order enough more made to replace all worn items. If there are enough replacement items in inventory to replace all worn items, the units wearing them will have the worn items confiscated (in the same manner as the *cleanowned* plugin) so that they'll reeequip with replacement items.

Use the *enable* and *disable* commands to toggle this plugin's status, or run `tailor status` to check its current status.

5.3.24 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
```

5.3.25 workNow

Don't allow dwarves to idle if any jobs are available.

When workNow is active, every time the game pauses, DF will make dwarves perform any appropriate available jobs. This includes when you one step through the game using the pause menu. Usage:

workNow print workNow status

workNow 0 deactivate workNow

workNow 1 activate workNow (look for jobs on pause, and only then)

workNow 2 make dwarves look for jobs whenever a job completes

5.3.26 zone

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

Keybinding: AltShiftI -> `"zone set"` in `dwarfmode/Zones`

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.

enumnick Assign enumerated nicknames (e.g. "Hen 1", "Hen 2"...). Must be followed by the prefix to use in 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.

5.4 Map modification

5.4.1 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.

5.4.2 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

5.4.3 burrows

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.

5.4.4 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.

5.4.5 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?

5.4.6 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.

5.4.7 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.

5.4.8 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).

5.4.9 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`.

Keybinding: CtrlV

Keybinding: CtrlShiftV -> `"digv x"`

Note: All commands implemented by the *dig* plugin (listed by `ls dig`) support specifying the designation priority with `-p#`, `-p #`, or `p=#`, where `#` is a number from 1 to 7. If a priority is not specified, the priority selected in-game is used as the default.

5.4.10 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.

5.4.11 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

5.4.12 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

5.4.13 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

5.4.14 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.

5.4.15 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 Tree: Select trees only (exclude shrubs)

-s Shrub: Select shrubs only (exclude trees)

-f Farming: Designate only shrubs that yield seeds for farming. Implies -s

-c Clear: Clear designations instead of setting them

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

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

-v Verbose: Lists the number of (un)designations per plant

-n * Number: Designate up to * (an integer number) plants of each species

Specifying both `-t` and `-s` or `-f` will have no effect. If no plant IDs are specified, all valid plant IDs will be listed, with `-t`, `-s`, and `-f` restricting the list to trees, shrubs, and farmable shrubs, respectively.

Note: DF is capable of determining that a shrub has already been picked, leaving an unusable structure part behind. This plugin does not perform such a check (as the location of the required information has not yet been identified). This leads to some shrubs being designated when they shouldn't be, causing a plant gatherer to walk there and do nothing (except clearing the designation). See Issue 1479 for details.

The implementation another known deficiency: it's incapable of detecting that raw definitions that specify a seed extraction reaction for the structural part but has no other use for it cannot actually yield any seeds, as the part is never used (parts of Bug 6940, e.g. Red Spinach), even though DF collects it, unless there's a workshop reaction to do it (which there isn't in vanilla).

5.4.16 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.

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

5.4.17 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.

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

Note: *gui/liquids* is an in-game UI for this script.

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.

5.4.18 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.

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!

5.4.19 regrass

Regrows all the grass. Not much to it ;)

5.4.20 restrictice

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

5.4.21 restrictliquids

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

5.4.22 tiletypes

Can be used for painting map tiles and is an interactive command, much like *liquids*. Some properties of existing tiles can be looked up with *probe*. 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.

Example:

```
tiletypes-command p any ; p s wall ; p sp normal
```

This resets the paint filter to unsmoothed walls.

tiletypes-here

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

Options:

-c, --cursor <x>,<y>,<z> Use the specified map coordinates instead of the current cursor position. If this option is specified, then an active game map cursor is not necessary.

-h, --help Show command help text.

-q, --quiet Suppress non-error status output.

tiletypes-here-point

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

This command supports the same options as *tiletypes-here* above.

5.4.23 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.

5.5 Mods and Cheating

5.5.1 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*.

5.5.2 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.

Keybinding: CtrlB in `dungeonmode`

Keybinding: CtrlShiftB -> `"adv-bodyswap force"` in `dungeonmode`

5.5.3 createitem

Allows creating new items of arbitrary types and made of arbitrary materials. A unit must be selected in-game to use this command. 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, the `:NONE` can be omitted
- If the item is `REMAINS`, `FISH`, `FISH_RAW`, `VERMIN`, `PET`, or `EGG`, specify a `CREATURE:CASTE` pair instead of a material token.
- If the item is a `PLANT_GROWTH`, specify a `PLANT_ID:GROWTH_ID` pair instead of a material token.

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

To obtain the item and material tokens of an existing item, run `createitem inspect`. Its output can be passed directly as arguments to `createitem` to create new matching items, as long as the item type is supported.

Examples:

- Create 2 pairs of steel gauntlets:

```
createitem GLOVES:ITEM_GLOVES_GAUNTLETS INORGANIC:STEEL 2
```

- Create tower-cap logs:

```
createitem WOOD PLANT_MAT:TOWER_CAP:WOOD
```

- Create bilberries:

```
createitem PLANT_GROWTH BILBERRY:FRUIT
```

For more examples, see this wiki page.

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).

5.5.4 dig-now

Instantly completes non-marker dig designations, modifying tile shapes and creating boulders, ores, and gems as if a miner were doing the mining or engraving. By default, the entire map is processed and boulder generation follows standard game rules, but the behavior is configurable.

Note that no units will get mining or engraving experience for the dug/engraved tiles.

Trees and roots are not currently handled by this plugin and will be skipped. Requests for engravings are also skipped since they would depend on the skill and creative choices of individual engravers. Other types of engraving (i.e. smoothing and track carving) are handled.

Usage:

```
dig-now [<pos> [<pos>]] [<options>]
```

Where the optional <pos> pair can be used to specify the coordinate bounds within which dig-now will operate. If they are not specified, dig-now will scan the entire map. If only one <pos> is specified, only the tile at that coordinate is processed.

Any <pos> parameters can either be an <x>,<y>,<z> triple (e.g. 35,12,150) or the string here, which means the position of the active game cursor should be used.

Examples:

dig-now Dig designated tiles according to standard game rules.

dig-now --clean Dig designated tiles, but don't generate any boulders, ores, or gems.

dig-now --dump here Dig tiles and dump all generated boulders, ores, and gems at the tile under the game cursor.

Options:

-c, --clean Don't generate any boulders, ores, or gems. Equivalent to --percentages 0,0,0,0.

-d, --dump <pos> Dump any generated items at the specified coordinates. If the tile at those coordinates is open space or is a wall, items will be generated on the closest walkable tile below.

-e, --everywhere Generate a boulder, ore, or gem for every tile that can produce one. Equivalent to --percentages 100,100,100,100.

-h, --help Show quick usage help text.

-p, --percentages <layer>,<vein>,<small cluster>,<deep> Set item generation percentages for each of the tile categories. The vein category includes both the large oval clusters and the long stringy mineral veins. Default is 25,33,100,100.

-z, --cur-zlevel Restricts the bounds to the currently visible z-level.

5.5.5 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

5.5.6 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.

5.5.7 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*.

5.5.8 generated-creature-renamer

Automatically renames generated creatures, such as forgotten beasts, titans, etc, to have raw token names that match the description given in-game.

The `list-generated` command can be used to list the token names of all generated creatures in a given save, with an optional `detailed` argument to show the accompanying description.

The `save-generated-raws` command will save a sample creature graphics file in the Dwarf Fortress root directory, to use as a start for making a graphics set for generated creatures using the new names that they get with this plugin.

The new names are saved with the save, and the plugin, when enabled, only runs once per save, unless there's an update.

5.5.9 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)

5.5.10 misery

When enabled, fake bad thoughts will be added to all dwarves.

Usage:

misery enable n enable misery with optional magnitude n. If specified, n must be positive.

misery n same as "misery enable n"

misery enable same as "misery enable 1"

misery disable stop adding new negative thoughts. This will not remove existing negative thoughts. Equivalent to "misery 0".

misery clear remove fake thoughts, even after saving and reloading. Does not change factor.

5.5.11 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.

5.5.12 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*.

5.5.13 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*.

5.5.14 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.

5.5.15 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 T are "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.

5.6 Plugin Lua API

Some plugins consist solely of native libraries exposed to Lua. They are listed in the *DFHack Lua API* file under *Plugins*:

- *building-hacks*
- *cxxrandom*
- *eventful*

- *luasocket*
- *map-render*
- *pathable*
- *xlsxreader*

5.7 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.

- *automaterial*
- *automelt*
- *autotrade*
- *buildingplan*
  - *Item filtering*
  - *Quickfort mode*
  - *Global settings*
- *command-prompt*
- *confirm*
- *debug*
  - *help*
  - *category*
  - *filter*
  - *set*
  - *unset*
  - *disable*
  - *enable*
- *embark-assistant*
- *embark-tools*
- *follow*
- *hotkeys*
- *manipulator*
  - *Professions*

- *mousequery*
- *nopause*
- *rename*
- *rendermax*
- *resume*
- *ruby*
- *search*
- *sort-items*
- *sort-units*
- *stockpiles*
- *stocks*
- *title-folder*
- *title-version*
- *trackstop*

5.7.1 automaterial

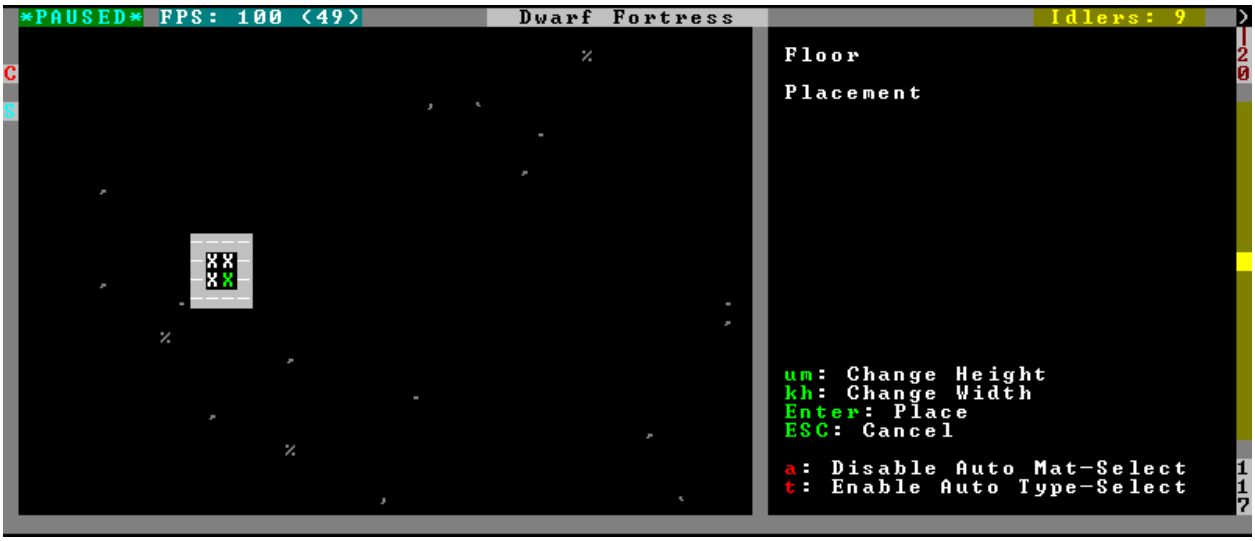
This makes building constructions (walls, floors, fortifications, etc) a little bit easier by saving you from having to trawl through long lists of materials each time you place one.

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.

5.7.2 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*.

5.7.3 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*.

5.7.4 buildingplan

When active (via `enable buildingplan`), this plugin adds a planning mode for building placement. You can then place furniture, constructions, and other buildings before the required materials are available, and they will be created in a suspended state. Buildingplan will periodically scan for appropriate items, and the jobs will be unsuspended when the items are available.

This is 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.

Item filtering

While placing a building, you can set filters for what materials you want the building made out of, what quality you want the component items to be, and whether you want the items to be decorated.

If a building type takes more than one item to construct, use `CtrlLeft` and `CtrlRight` to select the item that you want to set filters for. Any filters that you set will be used for all buildings of the selected type placed from that point onward (until you set a new filter or clear the current one). Buildings placed before the filters were changed will keep the filter values that were set when the building was placed.

For example, you can be sure that all your constructed walls are the same color by setting a filter to accept only certain types of stone.

Quickfort mode

If you use the external Python Quickfort to apply building blueprints instead of the native DFHack *quickfort* script, you must enable Quickfort mode. This temporarily enables buildingplan for all building types and adds an extra blank screen after every building placement. This "dummy" screen is needed for Python Quickfort to interact successfully with Dwarf Fortress.

Note that Quickfort mode is only for compatibility with the legacy Python Quickfort. The DFHack *quickfort* script does not need Quickfort mode to be enabled. The *quickfort* script will successfully integrate with buildingplan as long as the buildingplan plugin is enabled.

Global settings

The buildingplan plugin has several global settings that can be set from the UI (`G` from any building placement screen, for example: `baG`). These settings can also be set from the `DFHack#` prompt once a map is loaded (or from your `onMapLoad.init` file) with the syntax:

```
buildingplan set <setting> <true|false>
```

and displayed with:

```
buildingplan set
```

The available settings are:

Setting	Default	Persisted	Description
all_enabled	false	no	Enable planning mode for all building types.
blocks	true	yes	Allow blocks, boulders, logs, or bars to be matched for generic "building material" items
boulders	true		
logs	true		
bars	false		
quickfort_mode	false	no	Enable compatibility mode for the legacy Python Quickfort (not required for DFHack quickfort)

For example, to ensure you only use blocks when a "building material" item is required, you could add this to your `onMapLoad.init` file:

```
on-new-fortress buildingplan set boulders false; buildingplan set logs false
```

Persisted settings (i.e. `blocks`, `boulders`, `logs`, and `bars`) are saved with your game, so you only need to set them to the values you want once.

5.7.5 command-prompt

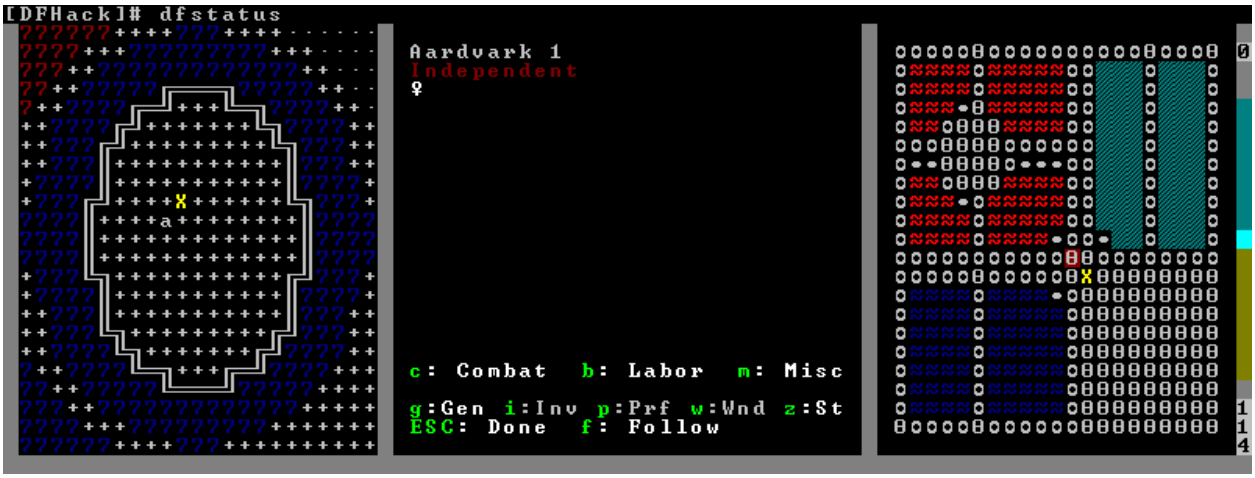
An in-game DFHack terminal, where you can enter other commands.

Keybinding: 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*.



5.7.6 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.

5.7.7 debug

Manager for DFHack runtime debug prints. Debug prints are grouped by plugin name, category name and print level. Levels are `trace`, `debug`, `info`, `warning` and `error`.

The runtime message printing is controlled using filters. Filters set the visible messages of all matching categories. Matching uses regular expression syntax, which allows listing multiple alternative matches or partial name matches. This syntax is a C++ version of the ECMA-262 grammar (Javascript regular expressions). Details of differences can be found at https://en.cppreference.com/w/cpp/regex/ecmascript

Persistent filters are stored in `dfhack-config/runtime-debug.json`. Oldest filters are applied first. That means a newer filter can override the older printing level selection.

Usage: `debugfilter [subcommand] [parameters...]`

The following subcommands are supported:

help

Give overall help or a detailed help for a subcommand.

Usage: `debugfilter help [subcommand]`

category

List available debug plugin and category names.

Usage: `debugfilter category [plugin regex] [category regex]`

The list can be filtered using optional regex parameters. If filters aren't given then the it uses `"."` regex which matches any character. The regex parameters are good way to test regex before passing them to `set`.

filter

List active and passive debug print level changes.

Usage: `debugfilter filter [id]`

Optional `id` parameter is the id listed as first column in the filter list. If id is given then the command shows information for the given filter only in multi line format that is better format if filter has long regex.

set

Creates a new debug filter to set category printing levels.

Usage: `debugfilter set [level] [plugin regex] [category regex]`

Adds a filter that will be deleted when DF process exists or plugin is unloaded.

Usage: `debugfilter set persistent [level] [plugin regex] [category regex]`

Stores the filter in the configuration file to until `unset` is used to remove it.

Level is the minimum debug printing level to show in log.

- `trace`: Possibly very noisy messages which can be printed many times per second
- `debug`: Messages that happen often but they should happen only a couple of times per second
- `info`: Important state changes that happen rarely during normal execution
- `warning`: Enabled by default. Shows warnings about unexpected events which code managed to handle correctly.
- `error`: Enabled by default. Shows errors which code can't handle without user intervention.

unset

Delete a space separated list of filters

Usage: `debugfilter unset [id...]`

disable

Disable a space separated list of filters but keep it in the filter list

Usage: `debugfilter disable [id...]`

enable

Enable a space sperate list of filters

Usage: `debugfilter enable [id...]`

5.7.8 embark-assistant

This plugin provides embark site selection help. It has to be run with the `embark-assistant` command while the pre-embark screen is displayed and shows extended (and correct(?)) resource information for the embark rectangle as well as normally undisplayed sites in the current embark region. It also has a site selection tool with more options than DF's vanilla search tool. For detailed help invoke the in game info screen.

5.7.9 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

5.7.10 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.

5.7.11 hotkeys

Opens an in-game screen showing which DFHack keybindings are active in the current context. See also *hotkey-notes*.

Keybinding: CtrlF1

Keybinding: AltF1

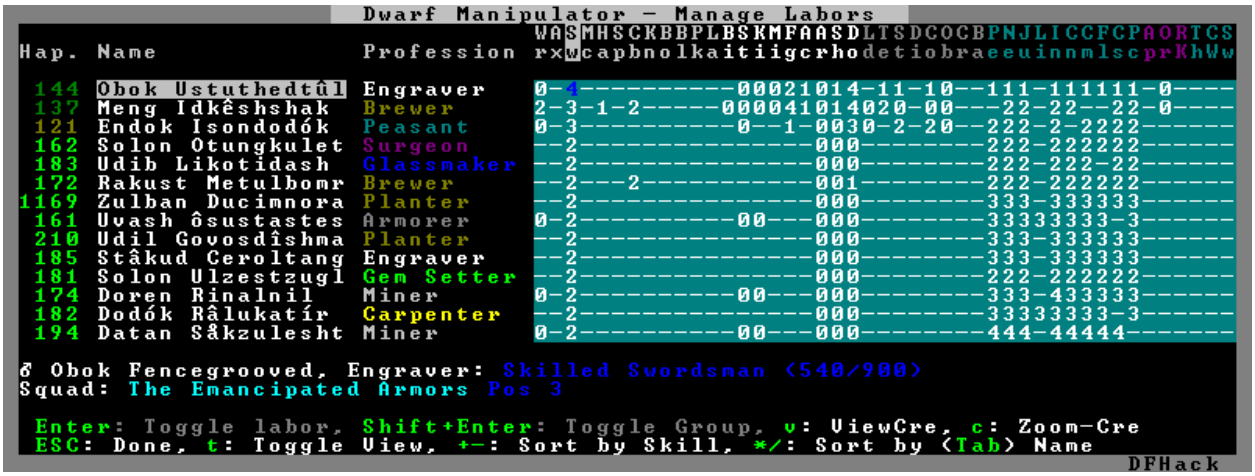
5.7.12 manipulator

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, Squad, and Job views.

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.

Professions

The manipulator plugin supports saving professions: a named set of labors that can be quickly applied to one or multiple dwarves.

To save a profession, highlight a dwarf and press `P`. The profession will be saved using the custom profession name of the dwarf, or the default for that dwarf if no custom profession name has been set.

To apply a profession, either highlight a single dwarf or select multiple with `x`, and press `p` to select the profession to apply. All labors for the selected dwarves will be reset to the labors of the chosen profession.

Professions are saved as human-readable text files in the "professions" folder within the DF folder, and can be edited or deleted there.

5.7.13 mousequery

Adds mouse controls to the DF interface, e.g. 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]
```

5.7.14 nopause

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

5.7.15 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.

5.7.16 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.

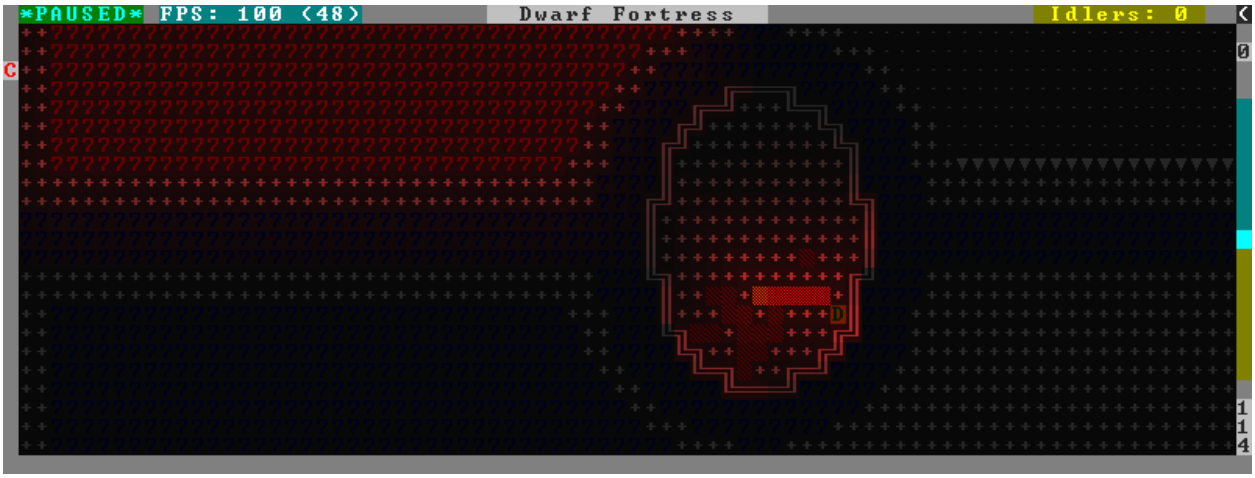
light reload Reload the settings file.

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...



5.7.17 resume

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

5.7.18 ruby

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`.

5.7.19 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.

5.7.20 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`

5.7.21 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`

Keybinding: AltShiftN -> `"sort-units name" "sort-items description"`

Keybinding: AltShiftR -> `"sort-units arrival"`

Keybinding: AltShiftT -> `"sort-units profession" "sort-items type material"`

Keybinding: AltShiftQ -> `"sort-units squad_position" "sort-items quality"`

5.7.22 stockpiles

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.

Keybinding: AltP

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. e.g.: `savestock food_settings.dfstock`

loadstock Loads a saved stockpile settings file and applies it to the currently selected stockpile. e.g.: `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.

5.7.23 stocks

Replaces the DF stocks screen with an improved version.

Keybinding: CtrlShiftZ -> `"stocks show"` in `dwarfmode/Default`

5.7.24 title-folder

Displays the DF folder name in the window title bar when enabled.

5.7.25 title-version

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

5.7.26 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.

CHAPTER 6

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.

6.1 Basic Scripts

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

Contents

- *embark-skills*
- *emigration*
- *empty-bin*
- *exportlegends*
- *exterminate*
- *extinguish*
- *feature*
- *fillneeds*
- *firestarter*
- *fix-ster*
- *fixnaked*
- *flashstep*
- *force*
- *forget-dead-body*
- *forum-dwarves*
- *full-heal*
- *gaydar*
- *geld*
- *ghostly*
- *growcrops*
- *hermit*
- *hfs-pit*
- *hotkey-notes*
- *install-info*
- *item-descriptions*
- *launch*
- *lever*
- *light-aquifers-only*
- *linger*
- *list-agreements*
- *list-waves*
- *load-save*
- *locate-ore*
- *lua*
- *make-legendary*

- *make-monarch*
- *markdown*
- *masspit*
- *migrants-now*
- *multicmd*
- *names*
- *on-new-fortress*
- *once-per-save*
- *open-legends*
- *points*
- *position*
- *pref-adjust*
- *prefchange*
- *prioritize*
- *putontable*
- *questport*
- *quickfort*
- *quicksave*
- *region-pops*
- *rejuvenate*
- *remove-stress*
- *remove-wear*
- *repeat*
- *resurrect-adv*
- *reveal-adv-map*
- *reveal-hidden-sites*
- *reveal-hidden-units*
- *season-palette*
- *set-orientation*
- *set-timeskip-duration*
- *setfps*
- *show-unit-syndromes*
- *siren*
- *source*
- *spawnunit*

6.1.1 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).

6.1.2 add-recipe

Adds unknown weapon and armor crafting recipes to your civ. E.g. some civilizations never learn to craft high boots. This script can help with that, and more. Only weapons, armor, and tools are currently supported; things such as instruments are not. Available options:

- `add-recipe all` adds *all* available weapons and armor, including exotic items like blowguns, two-handed swords, and capes.

- `add-recipe native` adds only native (but unknown) crafting recipes. Civilizations pick randomly from a pool of possible recipes, which means not all civs get high boots, for instance. This command gives you all the recipes your civilisation could have gotten.
- `add-recipe single <item token>` adds a single item by the given item token. For example:

```
add-recipe single SHOES:ITEM_SHOES_BOOTS
```

6.1.3 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.

6.1.4 adv-fix-sleepers

Fixes Bug 6798. This bug is characterized by sleeping units who refuse to awaken in adventure mode regardless of talking to them, hitting them, or waiting so long you die of thirst. If you come accross one or more bugged sleepers in adventure mode, simply run the script and all nearby sleepers will be cured.

Usage:

```
adv-fix-sleepers
```

6.1.5 adv-max-skills

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

6.1.6 adv-rumors

Improves the "Bring up specific incident or rumor" menu in Adventure mode.

- Moves entries into one line
- Adds a "slew" keyword for filtering, making it easy to find your kills and not your companions'
- Trims repetitive words

Keybinding: CtrlA in `dungeonmode/ConversationSpeak`

6.1.7 animal-control

Animal control is a script useful for deciding what animals to butcher and geld.

While not as powerful as Dwarf Therapist in managing animals - in so far as DT allows you to sort by various stats and flags - this script does provide many options for filtering animals. Additionally you can mark animals for slaughter or gelding, you can even do so enmasse if you so choose.

Examples:

```
animal-control -race DOG
animal-control -race DOG -male -notgelded -showstats
animal-control -markfor gelding -id 1988
animal-control -markfor slaughter -id 1988
animal-control -gelded -markedfor slaughter -unmarkfor slaughter
```

Selection options:

These options are used to specify what animals you want or do not want to select.

`-all`: Selects all units. Note: cannot be used in conjunction with other selection options.

`-id <value>`: Selects the unit with the specified id value provided.

`-race <value>`: Selects units which match the race value provided.

`-markedfor <action>`: Selects units which have been marked for the action provided. Valid actions: `slaughter`, `gelding`

`-notmarkedfor <action>`: Selects units which have not been marked for the action provided. Valid actions: `slaughter`, `gelding`

`-gelded`: Selects units which have already been gelded.

`-notgelded`: Selects units which have not been gelded.

`-male`: Selects units which are male.

`-female`: Selects units which are female.

Command options:

- `-showstats`: Displays physical attributes of the selected animals.
- **`-markfor <action>`: Marks selected animals for the action provided.** Valid actions: `slaughter`, `gelding`
- **`-unmarkfor <action>`: Unmarks selected animals for the action provided.** Valid actions: `slaughter`, `gelding`

Other options:

- `-help`: Displays this information

Column abbreviations

Due to space constraints, the names of some output columns are abbreviated as follows:

- `str`: strength
- `agi`: agility
- `tgh`: toughness
- `endur`: endurance
- `recup`: recuperation
- `disres`: disease resistance

6.1.8 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 and trait to an ideal easy-to-satisfy preference.

Without providing arguments, only attributes, age, and personalities will be adjusted. Adding arguments allows for skills or classes to be adjusted to legendary (maximum).

Arguments:

- **`list`** Prints list of all skills
- **`classes`** Prints list of all classes

- **all** Set all skills, for all Dwarves, to legendary
- **<skill name>** Set a specific skill, for all Dwarves, to legendary

  example: `armoks-blessing RANGED_COMBAT`

  All Dwarves become a Legendary Archer
- **<class name>** Set a specific class (group of skills), for all Dwarves, to legendary

  example: `armoks-blessing Medical`

  All Dwarves will have all medical related skills set to legendary

6.1.9 assign-attributes

A script to change the physical and mental attributes of a unit.

Attributes are divided into tiers from -4 to 4. Tier 0 is the standard level and represents the average values for that attribute, tier 4 is the maximum level, and tier -4 is the minimum level.

An example of the attribute "Strength":

Tier	Description
4	unbelievably strong
3	mighty
2	very strong
1	strong
0	(no description)
-1	weak
-2	very weak
-3	unquestionably weak
-4	unfathomably weak

For more information: https://dwarffortresswiki.org/index.php/DF2014:Attribute

Usage:

-help: print the help page.

-unit <UNIT_ID>: the target unit ID. If not present, the currently selected unit will be the target.

-attributes [<ATTRIBUTE> <TIER> <ATTRIBUTE> <TIER> <...>]: the list of the attributes to modify and their tiers. The valid attribute names can be found in the wiki: https://dwarffortresswiki.org/index.php/DF2014:Attribute (substitute any space with underscores); tiers range from -4 to 4. There must be a space before and after each square bracket.

-reset: reset all attributes to the average level (tier 0). If both this option and a list of attributes/tiers are present, the unit attributes will be reset and then the listed attributes will be modified.

Example:

```
assign-attributes -reset -attributes [ STRENGTH 2 AGILITY -1 SPATIAL_SENSE -1 ]
```

This will reset all attributes to a neutral value and will set the following values (if the currently selected unit is a dwarf):

- Strength: a random value between 1750 and 1999 (tier 2);
- Agility: a random value between 401 and 650 (tier -1);
- Spatial sense: a random value between 1043 and 1292 (tier -1).

The final result will be: "She is very strong, but she is clumsy. She has a questionable spatial sense."

6.1.10 assign-beliefs

A script to change the beliefs (values) of a unit.

Beliefs are defined with the belief token and a number from -3 to 3, which describes the different levels of belief strength, as explained here: https://dwarffortresswiki.org/index.php/DF2014:Personality_trait#Beliefs

Strength	Effect
3	Highest
2	Very High
1	High
0	Neutral
-1	Low
-2	Very Low
-3	Lowest

Resetting a belief means setting it to a level that does not trigger a report in the "Thoughts and preferences" screen.

Usage:

-help: print the help page.

-unit <UNIT_ID>: set the target unit ID. If not present, the currently selected unit will be the target.

-beliefs [<BELIEF> <LEVEL> <BELIEF> <LEVEL> <...>]: the beliefs to modify and their levels. The valid belief tokens can be found in the wiki page linked above; level values range from -3 to 3. There must be a space before and after each square bracket.

-reset: reset all beliefs to a neutral level. If the script is called with both this option and a list of beliefs/levels, first all the unit beliefs will be reset and then those beliefs listed after `-beliefs` will be modified.

Example:

```
assign-beliefs -reset -beliefs [ TRADITION 2 CRAFTSMANSHIP 3 POWER 0 CUNNING -1 ]
```

Resets all the unit beliefs, then sets the listed beliefs to the following values:

- Tradition: a random value between 26 and 40 (level 2);
- Craftsmanship: a random value between 41 and 50 (level 3);
- Power: a random value between -10 and 10 (level 0);
- Cunning: a random value between -25 and -11 (level -1).

The final result (for a dwarf) will be: "She personally is a firm believer in the value of tradition and sees guile and cunning as indirect and somewhat worthless."

Note that the beliefs aligned with the cultural values of the unit have not triggered a report.

6.1.11 assign-facets

A script to change the facets (traits) of a unit.

Facets are defined with a token and a number from -3 to 3, which describes the different levels of facets strength, as explained here: https://dwarffortresswiki.org/index.php/DF2014:Personality_trait#Facets

Strength	Effect
3	Highest
2	Very High
1	High
0	Neutral
-1	Low
-2	Very Low
-3	Lowest

Resetting a facet means setting it to a level that does not trigger a report in the "Thoughts and preferences" screen.

Usage:

-help: print the help page.

-unit <UNIT_ID>: set the target unit ID. If not present, the currently selected unit will be the target.

-beliefs [<FACET> <LEVEL> <FACET> <LEVEL> <...>]: the facets to modify and their levels. The valid facet tokens can be found in the wiki page linked above; level values range from -3 to 3. There must be a space before and after each square bracket.

-reset: reset all facets to a neutral level. If the script is called with both this option and a list of facets/levels, first all the unit facets will be reset and then those facets listed after `-facets` will be modified.

Example:

```
assign-facets -reset -facets [ HATE_PROPENSITY -2 CHEER_PROPENSITY -1 ]
```

Resets all the unit facets, then sets the listed facets to the following values:

- Hate propensity: a value between 10 and 24 (level -2);
- Cheer propensity: a value between 25 and 39 (level -1).

The final result (for a dwarf) will be: "She very rarely develops negative feelings toward things. She is rarely happy or enthusiastic, and she is conflicted by this as she values parties and merrymaking in the abstract."

Note that the facets are compared to the beliefs, and if conflicts arise they will be reported.

6.1.12 assign-goals

A script to change the goals (dreams) of a unit.

Goals are defined with the goal token and a true/false value that describes whether or not the goal has been accomplished. Be advised that this last feature has not been properly tested and might be potentially destructive: I suggest leaving it at false.

For a list of possible goals: https://dwarffortresswiki.org/index.php/DF2014:Personality_trait#Goals

Bear in mind that nothing will stop you from assigning zero or more than one goal, but it's not clear how it will affect the game.

Usage:

-help: print the help page.

-unit <UNIT_ID>: set the target unit ID. If not present, the currently selected unit will be the target.

-goals [<GOAL> <REALIZED_FLAG> <GOAL> <REALIZED_FLAG> <...>]: the goals to modify/add and whether they have been realized or not. The valid goal tokens can be found in the wiki page linked above. The flag must be a true/false value. There must be a space before and after each square bracket.

-reset: clear all goals. If the script is called with both this option and a list of goals, first all the unit goals will be erased and then those goals listed after `-goals` will be added.

Example:

```
assign-goals -reset -goals [ MASTER_A_SKILL false ]
```

Clears all the unit goals, then sets the "master a skill" goal. The final result will be: "dreams of mastering a skill".

6.1.13 assign-preferences

A script to change the preferences of a unit.

Preferences are classified into 12 types. The first 9 are:

- like material;
- like creature;
- like food;
- hate creature;
- like item;
- like plant;
- like tree;
- like colour;
- like shape.

These can be changed using this script.

The remaining three are not currently managed by this script, and are: like poetic form, like musical form, like dance form.

To produce the correct description in the "thoughts and preferences" page, you must specify the particular type of preference. For each type, a description is provided in the section below.

You will need to know the token of the object you want your dwarf to like. You can find them in the wiki, otherwise in the folder "/raw/objects/" under the main DF directory you will find all the raws defined in the game.

For more information: https://dwarffortresswiki.org/index.php/DF2014:Preferences

Usage:

-help: print the help page.

-unit <UNIT_ID>: set the target unit ID. If not present, the currently selected unit will be the target.

-likematerial [<TOKEN> <TOKEN> <...>]: usually a type of stone, a type of metal and a type of gem, plus it can also be a type of wood, a type of glass, a type of leather, a type of horn, a type of pearl, a type of ivory, a decoration material - coral or amber, a type of bone, a type of shell, a type of silk, a type of yarn, or a type of plant cloth. Write the full tokens. There must be a space before and after each square bracket.

-likecreature [<TOKEN> <TOKEN> <...>]: one or more creatures liked by the unit. You can just list the species: the creature token will be something similar to `CREATURE:SPARROW:SKIN`, so the name of the species will be `SPARROW`. Nothing will stop you to write the full token, if you want: the script will just ignore the first and the last parts. There must be a space before and after each square bracket.

-likefood [<TOKEN> <TOKEN> <...>]: usually a type of alcohol, plus it can be a type of meat, a type of fish, a type of cheese, a type of edible plant, a cookable plant/creature extract, a cookable mill powder, a cookable plant seed or a cookable plant leaf. Write the full tokens. There must be a space before and after each square bracket.

-hatecreature [<TOKEN> <TOKEN> <...>]: works the same way as `-likecreature`, but this time it's one or more creatures that the unit detests. They should be a type of `HATEABLE` vermin which isn't already explicitly liked, but no check is performed about this. Like before, you can just list the creature species. There must be a space before and after each square bracket.

-likeitem [<TOKEN> <TOKEN> <...>]: a kind of weapon, a kind of ammo, a kind of piece of armor, a piece of clothing (including backpacks or quivers), a type of furniture (doors, floodgates, beds, chairs, windows, cages, barrels, tables, coffins, statues, boxes, armor stands, weapon racks, cabinets, bins, hatch covers, grates, querns, millstones, traction benches, or slabs), a kind of craft (figurines, amulets, scepters, crowns, rings, earrings, bracelets, or large gems), or a kind of miscellaneous item (catapult parts, ballista parts, a type of siege ammo, a trap component, coins, anvils, totems, chains, flasks, goblets, buckets, animal traps, an instrument, a toy, splints, crutches, or a tool). The item tokens can be found here: https://dwarffortresswiki.org/index.php/DF2014:Item_token If you want to specify an item subtype, look into the files listed under the column "Subtype" of the wiki page (they are in the "/raw/ojects/" folder), then specify the items using the full tokens found in those files (see examples below). There must be a space before and after each square bracket.

-likeplant [<TOKEN> <TOKEN> <...>]: works in a similar way as `-likecreature`, this time with plants. You can just List the plant species (the middle part of the token). There must be a space before and after each square bracket.

-liketree [<TOKEN> <TOKEN> <...>]: works exactly as `-likeplant`. I think this preference type is here for backward compatibility (?). You can still use it, however. As before, you can just list the tree (plant) species. There must be a space before and after each square bracket.

-likecolor [<TOKEN> <TOKEN> <...>]: you can find the color tokens here: https://dwarffortresswiki.org/index.php/DF2014:Color#Color_tokens or inside the "descriptor_color_standard.txt" file (in the "/raw/ojects/" folder). You can use the full token or just the color name. There must be a space before and after each square bracket.

-likeshape [<TOKEN> <TOKEN> <...>]: I couldn't find a list of shape tokens in the wiki, but you can find them inside the "descriptor_shape_standard.txt" file (in the "/raw/ojects/" folder). You can use the full token or just the shape name. There must be a space before and after each square bracket.

-reset: clear all preferences. If the script is called with both this option and one or more preferences, first all the unit preferences will be cleared and then the listed preferences will be added.

Examples:

- "likes alabaster and willow wood":

```
assign-preferences -reset -likematerial [ INORGANIC:ALABASTER PLANT:WILLOW:WOOD ]
```

- "likes sparrows for their . . . ":

```
assign-preferences -reset -likecreature SPARROW
```

- "prefers to consume dwarven wine and olives":

```
assign-preferences -reset -likefood [ PLANT:MUSHROOM_HELMET_PLUMP:DRINK␣
↪PLANT:OLIVE:FRUIT ]
```

- "absolutely detests jumping spiders:

```
assign-preferences -reset -hatecreature SPIDER_JUMPING
```

- "likes logs and battle axes":

```
assign-preferences -reset -likeitem [ WOOD ITEM_WEAPON:ITEM_WEAPON_AXE_BATTLE ]
```

- "likes straberry plants for their ...":

```
assign-preferences -reset -likeplant BERRIES_STRAW
```

- "likes oaks for their ...":

```
assign-preferences -reset -liketree OAK
```

- "likes the color aqua":

```
assign-preferences -reset -likecolor AQUA
```

- "likes stars":

```
assign-preferences -reset -likeshape STAR
```

6.1.14 assign-profile

A script to change the characteristics of a unit according to a profile loaded from a json file.

A profile can describe which attributes, skills, preferences, beliefs, goals and facets a unit must have. The script relies on the presence of the other `assign-...` modules in this collection: please refer to the other modules documentation for more specific information.

For information about the json schema, please see the the "/hack/scripts/dwarf_profiles.json" file.

Usage:

-help: print the help page.

-unit <UNIT_ID>: the target unit ID. If not present, the target will be the currently selected unit.

-file <filename>: the json file containing the profile to apply. It's a relative path, starting from the DF root directory and ending at the json file. It must begin with a slash. Default value: "/hack/scripts/dwarf_profiles.json".

-profile <profile>: the profile to apply. It's the name of the profile as stated in the json file.

-reset [<list of characteristics>]: the characteristics to be reset/cleared. If not present, it will not clear or reset any characteristic, and it will simply add what is described in the profile. If it's a valid list of characteristic, those characteristics will be reset, and then what is described in the profile will be applied. If set to `PROFILE`, it will reset only the characteristics directly modified by the profile (and then the new values described will be applied). If set to `ALL`, it will reset EVERY characteristic and then it will apply the profile. Accepted values: `ALL`, `PROFILE`, `ATTRIBUTES`, `SKILLS`, `PREFERENCES`, `BELIEFS`, `GOALS`, `FACETS`. There must be a space before and after each square bracket. If only one value is provided, the square brackets can be omitted.

Examples:

- Loads and applies the profile called "DOCTOR" in the default json file, resetting/clearing all the characteristics changed by the profile, leaving the others unchanged, and then applies the new values:

```
assign-profile -profile DOCTOR -reset PROFILE
```

- Loads and applies the profile called "ARCHER" in the provided (fictional) json, keeping all the old characteristics but the attributes and the skills, which will be reset (and then, if the profile provides some attributes or skills values, those new values will be applied):

```
assign-profile -file /hack/scripts/military_profiles.json -profile ARCHER -reset␣
↪[ ATTRIBUTES SKILLS ]
```

6.1.15 assign-skills

A script to change the skills of a unit.

Skills are defined by their token and their rank. Skills tokens can be found here: https://dwarffortresswiki.org/index.php/DF2014:Skill_token

Below you can find a list of the first 16 ranks:

Rank	Skill name
0	Dabbling
1	Novice
2	Adequate
3	Competent
4	Skilled
5	Proficient
6	Talented
7	Adept
8	Expert
9	Professional
10	Accomplished
11	Great
12	Master
13	High Master
14	Grand Master
15+	Legendary

For more information: https://dwarffortresswiki.org/index.php/DF2014:Skill#Skill_level_names

Usage:

-help: print the help page.

-unit <UNIT_ID>: the target unit ID. If not present, the currently selected unit will be the target.

-skills [<SKILL> <RANK> <SKILL> <RANK> <...>]: the list of the skills to modify and their ranks. Rank values range from -1 (the skill is not learned) to normally 20 (legendary + 5). It is actually possible to go beyond 20, no check is performed. There must be a space before and after each square bracket.

-reset: clear all skills. If the script is called with both this option and a list of skills/ranks, first all the unit skills will be cleared and then the listed skills will be added.

Example:

```
assign-skills -reset -skills [ WOODCUTTING 3 AXE 2 ]
```

Clears all the unit skills, then adds the Wood cutter skill (competent level) and the Axeman skill (adequate level).

6.1.16 autolabor-artisans

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

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

6.1.17 autonick

Gives dwarves unique nicknames chosen randomly from `dfhack-config/autonick.txt`.

One nickname per line. Empty lines, lines beginning with # and repeat entries are discarded.

Dwarves with manually set nicknames are ignored.

If there are fewer available nicknames than dwarves, the remaining dwarves will go un-nicknamed.

You may wish to use this script with the "repeat" command, e.g: `repeat -name autonick -time 3 -timeUnits months -command [ autonick all ]`

Usage:

    autonick all [<options>] autonick help

Options:

-h, --help Show this text.

-q, --quiet Do not report how many dwarves were given nicknames.

6.1.18 autounsuspend

Periodically check construction jobs and keep them unsuspended with the *unsuspend* script.

6.1.19 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`, `fruit`, `mill`, `thread`, and `milk`.

6.1.20 binpatch

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

6.1.21 bodyswap

This script allows the player to take direct control of any unit present in adventure mode whilst giving up control of their current player character.

To specify the target unit, simply select it in the user interface, such as by opening the unit's status screen or viewing its description, and enter "bodyswap" in the DFHack console.

Alternatively, the target unit can be specified by its unit id as shown below.

Arguments:

```
-unit id
    replace "id" with the unit id of your target
    example:
        bodyswap -unit 42
```

6.1.22 brainwash

Modify the personality traits of the selected dwarf to match an idealised personality - for example, as stable and reliable as possible to prevent tantrums even after months of misery.

Usage: `brainwash <type>`, with one of the following types:

ideal reliable, with generally positive personality traits

baseline reset all personality traits to the average

stepford amplifies all good qualities to an excessive degree

wrecked amplifies all bad qualities to an excessive degree

6.1.23 break-dance

Breaks up dances, such as broken or otherwise hung dances that occur when a unit can't find a partner.

6.1.24 build-now

Instantly completes unsuspended building construction jobs. By default, all buildings on the map are completed, but the area of effect is configurable.

Note that no units will get architecture experience for any buildings that require that skill to construct.

Usage:

```
build-now [<pos> [<pos>]] [<options>]
```

Where the optional `<pos>` pair can be used to specify the coordinate bounds within which `build-now` will operate. If they are not specified, `build-now` will scan the entire map. If only one `<pos>` is specified, only the building at that coordinate is built.

The `<pos>` parameters can either be an `<x>,<y>,<z>` triple (e.g. `35,12,150`) or the string `here`, which means the position of the active game cursor.

Examples:

`build-now` Completes all unsuspended construction jobs on the map.

`build-now here` Builds the unsuspended, unconstructed building under the cursor.

Options:

`-h`, `--help` Show help text.

`-q`, `--quiet` Suppress informational output (error messages are still printed).

6.1.25 burial

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

6.1.26 cannibalism

Allows consumption of sapient corpses. Use from an adventurer's inventory screen or an individual item's detail screen.

6.1.27 caravan

Adjusts properties of caravans on the map. See also *force* to create caravans.

This script has multiple subcommands. Commands listed with the argument `[IDS]` can take multiple caravan IDs (see `caravan list`). If no IDs are specified, then the commands apply to all caravans on the map.

Subcommands:

- `list`: lists IDs and information about all caravans on the map.
- `extend [DAYS] [IDS]`: extends the time that caravans stay at the depot by the specified number of days (defaults to 7 if not specified). Also causes caravans to return to the depot if applicable.
- `happy [IDS]`: makes caravans willing to trade again (after seizing goods, annoying merchants, etc.). Also causes caravans to return to the depot if applicable.
- `leave [IDS]`: makes caravans pack up and leave immediately.

6.1.28 catsplosion

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

Usage:

catsplosion Make all cats pregnant

catsplosion list List IDs of all animals on the map

catsplosion ID ... Make animals with given ID(s) pregnant

Animals will give birth within two in-game hours (100 ticks or fewer).

6.1.29 clear-smoke

Removes all smoke from the map. Note that this can leak memory and should be used sparingly.

6.1.30 clear-webs

This script removes all webs that are currently on the map, and also frees any creatures who have been caught in one.

Note that it does not affect sprayed webs until they settle on the ground.

Usable in both fortress and adventurer mode.

Web removal and unit release happen together by default. The following may be used to isolate one of these actions:

Arguments:

```
-unitsOnly
    Include this if you want to free all units from webs
    without removing any webs

-websOnly
    Include this if you want to remove all webs
    without freeing any units
```

See also *fix/drop-webs*.

6.1.31 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.

6.1.32 color-schemes

A script to manage color schemes.

Current features are :

- **Registration**
- **Loading**
- **Listing**
- **Default** Setting/Loading

For detailed information and usage, type `color-schemes` in console.

Loaded as a module, this script will export those methods :

- register(path, force) : Register colors schemes by path (file or directory), relative to DF main directory
- load(name) : Load a registered color scheme by name
- list() : Return a list of registered color schemes
- set_default(name) : Set the default color scheme
- load_default() : Load the default color scheme

For more information about arguments and return values, see `hack/scripts/color-schemes.lua`.

Related scripts:

- *gui/color-schemes* is the in-game GUI for this script.
- *season-palette* swaps color schemes when the season changes.

6.1.33 combat-harden

Sets the combat-hardened value on a unit, making them care more/less about seeing corpses. Requires a value and a target.

Valid values:

-value <0-100> A percent value to set combat hardened to.

-tier <1-4>

Choose a tier of hardenedness to set it to. 1 = No hardenedness. 2 = "is getting used to tragedy" 3 = "is a hardened individual" 4 = "doesn't really care about anything anymore" (max)

If neither are provided, the script defaults to using a value of 100.

Valid targets:

-all All active units will be affected.

-citizens All (sane) citizens of your fort will be affected. Will do nothing in adventure mode.

-unit <UNIT ID> The given unit will be affected.

If no target is given, the provided unit can't be found, or no unit id is given with the unit argument, the script will try and default to targeting the currently selected unit.

6.1.34 combine-drinks

Merge stacks of drinks in the selected stockpile.

6.1.35 combine-plants

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

6.1.36 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 optional 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
```

6.1.37 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.

6.1.38 deep-embark

Moves the starting units and equipment to a specific underground region upon embarking.

This script can be run directly from the console at any point whilst setting up an embark.

Alternatively, create a file called "onLoad.init" in the DF raw folder (if one does not exist already) and enter the script command within it. Doing so will cause the script to run automatically and should hence be especially useful for modders who want their mod to include underground embarks by default.

Example:

```
deep-embark -depth CAVERN_2
```

Usage:

```
-depth X
    (obligatory)
    replace "X" with one of the following:
        CAVERN_1
        CAVERN_2
        CAVERN_3
        UNDERWORLD

-blockDemons
    including this arg will prevent demon surges
    in the context of breached underworld spires
    (intended mainly for UNDERWORLD embarks)
    ("wildlife" demon spawning will be unaffected)

-atReclaim
    if the script is being run from onLoad.init,
    including this arg will enable deep embarks
    when reclaiming sites too
    (there's no need to specify this if running
    the script directly from the console)

-clear
    re-enable normal surface embarks
```

6.1.39 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
```

6.1.40 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
```

6.1.41 deterioratefood

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
```

6.1.42 do-job-now

The script will try its best to find a job related to the selected entity (which can be a job, dwarf, animal, item, building, plant or work order) and then mark this job as high priority. There is no visual indicator, please look at the dfhack console for output. If a work order is selected, every job currently present job from this work order is affected, but not the future ones.

For best experience add the following to your `dfhack*.init`:

```
keybinding add Alt-N do-job-now
```

Also see `do-job-now` *tweak* and *prioritize*.

6.1.43 dorf_tables

Data tables for *dwarf-op*

Arguments:

- `-list [jobs|professions|types]`

Examples:

```
dorf_tables -list all
dorf_tables -list job_distributions
dorf_tables -list attrib_levels
dorf_tables -list jobs
dorf_tables -list professions
dorf_tables -list types
```

The data tables defined are described below.

job_distributions: Defines thresholds for each column of distributions. The columns must add up to the values in the thresholds row for that column. Every other row references an entry in 'jobs'

attrib_levels: Defines stat distributions, used for both physical and mental attributes. Each level gives a probability of a dwarf randomly being assigned an attribute level, and it provides a mean and standard deviation for the attribute's value.

jobs: Defines *dwarf-op*'s nameable jobs. This is preferable to taking every profession and making a distribution that covers all 100+ profs.

Each job is comprised of required professions, optional professions, probabilities for each optional profession, a 'max' number of optional professions, and a list of type(s) to apply to dwarves in the defined job.

professions: These are a subset of the professions DF has. All professions listed will match a profession dwarf fortress has built in, however not all the built-ins are defined here.

Each profession is defined with a set of job skills, these match the skills built in to dwarf fortress. Each skill is given a value which represents the bonus a dwarf will get to this skill. The skills are added in a random order, with the first few receiving the highest values (excluding the bonus just mentioned). Thus the bonuses are to ensure a minimum threshold is passed for certain skills deemed critical to a profession.

types: These are a sort of archetype system for applying to dwarves. It primarily includes physical attributes, but can include skills as well.

Each type has a probability of being applied to a dwarf just by pure luck - this is in addition to types applied by other means. Each type also has a list of attribute(s) each attribute has a attribute_level associated to it. Additionally each type may define a list of skills from the aforementioned job_skill listing, each skill is defined with a minimum value and maximum value, the given value is an evening distributed random number between these two numbers (inclusive).

To see a full list of built-in professions and jobs, you can run these commands:

```
devel/query -table df.profession
devel/query -table df.job_skill
```

6.1.44 drain-aquifer

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

6.1.45 dwarf-op

Dwarf optimization is a script designed to provide a robust solution to hacking dwarves to be better at work. The primary use case is as follows:

1) take a dwarf
2) delete their ability to do anything, even walk (job skills, phyiscal/mental attributes)
3) load the job distribution table from dorf_tables
4) update values in said table so the table accurately represents the distribution of your dwarves
5) pick an under-represented job from the table
6) apply the job to the dwarf, which means:
    - apply professions
    - provide custom profession name
    - add job skills
    - apply dwarf types
    - etc.

Beyond this use case of optimizing dwarves according to the tables in *dorf_tables*, this script makes each step in the process available to use separately, if you so choose.

There are two basic steps to using this script: selecting a subset of your dwarves, and running commands on those dwarves.

Usage:

```
dwarf-op -help
dwarf-op -select <select-option> -<command> <args>
```

Examples:

```
dwarf-op -select [ jobs Trader Miner Leader Rancher ] -applytype adaptable
dwarf-op -select all -clear -optimize
dwarf-op -select pall -clear -optimize
dwarf-op -select optimized -reroll
dwarf-op -select named -reroll inclusive -applyprofession RECRUIT
```

Select options:

Note: Prepend the letter `p` to any option to include protected dwarves in your selection

---

(none) same as typing '-select highlighted'

all selects all dwarves.

highlighted selects only the in-game highlighted dwarf (from any screen). [Ignores protection status]

<name> selects any dwarf with <name> in their name or nickname. (sub-string match) [Ignores protection status]

named selects dwarves with user-given names.

unnamed selects dwarves without user-given names.

employed selects dwarves with custom professions. Excludes optimized dwarves.

optimized selects dwarves based on session data. Dwarves who have been optimized should be listed in this data.

unoptimized selects any dwarves that don't appear in session data.

protected selects any dwarves which use protection signals in their name or profession. (i.e. `.`, `,`)

unprotected selects any dwarves which don't use protection signals in their name or profession.

drunks selects any dwarves which are currently zeroed, or were originally drunks as their profession.

jobs selects any dwarves with the listed jobs. This will only match with custom professions, or optimized dwarves (for optimized dwarves see jobs in *dorf_tables*).

Usage:

```
dwarf-op -select [ jobs job1 job2 etc. ]
```

Example:

```
dwarf-op -select [ jobs Miner Trader ]
```

waves selects dwarves from the specified migration waves. Waves are enumerated starting at 0 and increasing by 1 with each wave. The waves go by season and year and thus should match what you see in *list-waves* or Dwarf Therapist. It is recommended that you `-show` the selected dwarves before modifying.

Example:

```
dwarf-op -select [ waves 0 1 3 5 7 13 ]
```

General commands:

- `-reset`: deletes json file containing session data (bug: might not delete session data)
- `-resetall`: deletes both json files. session data and existing persistent data (bug: might not delete session data)
- `-show`: displays affected dwarves (id, name, migration wave, primary job). Useful for previewing selected dwarves before modifying them, or looking up the migration wave number for a group of dwarves.

Dwarf commands:

`clean <value>`: Cleans selected dwarves. Checks for skills with a rating of `<value>` and deletes them from the dwarf's skill list

`-clear`: Zeroes selected dwarves, or zeroes all dwarves if no selection is given. No attributes, no labours. Assigns `DRUNK` profession.

`-reroll [inclusive]`: zeroes selected dwarves, then rerolls that dwarf based on its job.

- Ignores dwarves with unlisted jobs.
- optional argument: `inclusive` - means your dorf(s) get the best of N rolls.
- See attrib_levels table in *dorf_tables* for `p` values describing the normal distribution of stats (each p value has a sub-distribution, which makes the bell curve not so bell-shaped). Labours do not follow the same stat system and are more uniformly random, which are compensated for in the description of jobs/professions.

`-optimize`: Performs a job search for unoptimized dwarves. Each dwarf will be found a job according to the job_distribution table in *dorf_tables*.

`-applyjobs`: Applies the listed jobs to the selected dwarves.

- List format: `[ job1 job2 jobn ]` (brackets and jobs all separated by spaces)
- See jobs table in *dorf_tables* for available jobs."

`-applyprofessions`: Applies the listed professions to the selected dwarves.

- List format: `[ prof1 prof2 profn ]` (brackets and professions all separated by spaces)
- See professions table in *dorf_tables* for available professions.

`-applytypes`: Applies the listed types to the selected dwarves.

- List format: `[ type1 type2 typen ]` (brackets and types all separated by spaces)
- See dwf_types table in *dorf_tables* for available types.

`renamejob <name>`: Renames the selected dwarves' custom profession to whatever is specified

Other Arguments:

`-help`: displays this help information.

`-debug`: enables debugging print lines

6.1.46 elevate-mental

Set all mental attributes of the selected dwarf to the maximum possible, or any number numbers between 0 and 5000 passed as an argument: `elevate-mental 100` for example would make the dwarf very stupid indeed.

6.1.47 elevate-physical

Set all physical attributes of the selected dwarf to the maximum possible, or any number numbers between 0 and 5000 passed as an argument. Higher is usually better, but an ineffective hammerer can be useful too...

6.1.48 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".

6.1.49 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
```

6.1.50 empty-bin

Empties the contents of the selected bin onto the floor.

6.1.51 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.

Usage:

```
exportlegends OPTION [FOLDER_NAME]
```

Valid values for `OPTION` are:

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

`FOLDER_NAME`, if specified, is the name of the folder where all the files will be saved. This defaults to the `legends-regionX-YYYYY-MM-DD` format. A path is also allowed, although everything but the last folder has to exist. To export to the top-level DF folder, pass `.` for this argument.

Examples:

- Export all information to the `legends-regionX-YYYYY-MM-DD` folder:

```
exportlegends all
```

- Export all information to the `region6` folder:

```
exportlegends all region6
```

- Export just the files included in `info` (above) to the `legends-regionX-YYYYY-MM-DD` folder:

```
exportlegends info
```

- Export just the custom XML file to the DF folder (no subfolder):

```
exportlegends custom .
```

Keybinding: CtrlA -> `"exportlegends all"` in `legends`

6.1.52 exterminate

Kills any unit of a given race.

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

With the special argument `this`, targets only the selected creature. Alternatively, `him`, `her`, `it`, `target`, and `selected` do the same thing.

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
exterminate gob:enemy
```

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

```
exterminate this
```

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

```
exterminate elve magma
```

6.1.53 extinguish

This script allows the user to put out fires affecting map tiles, plants, units, items and buildings.

To select a target, place the cursor over it before running the script. Alternatively, use one of the arguments below.

Arguments:

```
-all
    extinguish everything on the map

-location [ x y z ]
    extinguish only at the specified coordinates
```

6.1.54 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.

6.1.55 fillneeds

Use with a unit selected to make them focused and unstressed.

Alternatively, a unit can be specified by passing `-unit UNIT_ID`

Use `-all` to apply to all units on the map.

6.1.56 firestarter

Lights things on fire: items, locations, entire inventories even! Use while viewing an item, unit inventory, or tile to start fires.

6.1.57 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.

6.1.58 fixnaked

Removes all unhappy thoughts due to lack of clothing.

6.1.59 flashstep

A hotkey-friendly teleport that places your adventurer where your cursor is.

6.1.60 force

A simpler wrapper around the *modtools/force* script.

Usage:

- `force event_type`
- `force event_type civ_id` - civ ID required for `Diplomat` and `Caravan` events

See *modtools/force* for a complete list of event types.

6.1.61 forget-dead-body

Removes emotions associated with seeing a dead body.

6.1.62 forum-dwarves

Saves a copy of a text screen, formatted in bbcode for posting to the Bay12 Forums. See *markdown* to export for Reddit etc.

This script will attempt to read the current df-screen, and if it is a text-viewscreen (such as the dwarf 'thoughts' screen or an item 'description') then append a marked-up version of this text to the target file. Previous entries in the file are not overwritten, so you may use the 'forumdwarves' command multiple times to create a single document containing the text from multiple screens (eg: text screens from several dwarves, or text screens from multiple artifacts/items, or some combination).

The screens which have been tested and known to function properly with this script are:

1. dwarf/unit 'thoughts' screen
2. item/art 'description' screen
3. individual 'historical item/figure' screens

There may be other screens to which the script applies. It should be safe to attempt running the script with any screen active, with an error message to inform you when the selected screen is not appropriate for this script.

The target file's name is 'forumdwarves.txt'. A reminder to this effect will be displayed if the script is successful.

Note: The text will be encoded in CP437, which is likely to be incompatible with the system default. This causes incorrect display of special characters (eg. *é õ ç* = `é õ ç`). You can fix this by opening the file in an editor such as Notepad++ and selecting the correct encoding before using the text.

Keybinding: CtrlShiftF in `dwarfmode`

6.1.63 full-heal

Attempts to fully heal the selected unit from anything, optionally including death. Usage:

full-heal Completely heal the currently selected unit.

full-heal -unit [unitId] Apply command to the unit with the given ID, instead of selected unit.

full-heal -r [-keep_corpse] Heal the unit, raising from the dead if needed. Add `-keep_corpse` to avoid removing their corpse. The unit can be targeted by selecting its corpse on the UI.

full-heal -all [-r] [-keep_corpse] Heal all units on the map.

full-heal -all_citizens [-r] [-keep_corpse] Heal all fortress citizens on the map. Does not include pets.

full-heal -all_civ [-r] [-keep_corpse] Heal all units belonging to your parent civilisation, including pets and visitors.

For example, `full-heal -r -keep_corpse -unit ID_NUM` will fully heal unit ID_NUM. If this unit was dead, it will be resurrected without deleting the corpse - creepy!

6.1.64 gaydar

Shows the sexual orientation of units, useful for social engineering or checking the viability of livestock breeding programs.

Targets:

-all shows orientation of every creature

-citizens shows only orientation of citizens in fort mode

-named shows orientation of all named units on map

(no target) shows orientation of the unit under the cursor

Orientation filters:

-notStraight only creatures who are not strictly straight

-gayOnly only creatures who are strictly gay

-biOnly only creatures who can get into romances with both sexes

-straightOnly only creatures who are strictly straight

-asexualOnly only creatures who are strictly asexual

6.1.65 geld

Geld allows the user to geld and ungeld animals.

Valid options:

`-unit <id>`: Gelds the unit with the specified ID. This is optional; if not specified, the selected unit is used instead.

`-ungeld`: Ungelds the specified unit instead (see also *ungeld*).

`-toggle`: Toggles the gelded status of the specified unit.

`-help`: Shows this help information

6.1.66 ghostly

Toggles an adventurer's ghost status. Useful for walking through walls, avoiding attacks, or recovering after a death.

6.1.67 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 specified seeds, ready to be harvested.

Arguments:

- **`list` or `help`** List the various seed types currently in use in your farming plots
- **`<crop name>`** Grow specific planted seed type
- **`all`** Grow all planted seeds

Example:

- **Grow plump helmet spawn:** `growcrops plump`

6.1.68 hermit

Blocks all caravans, migrants, diplomats, and forgotten beasts (not wildlife). Useful for attempting the hermit challenge.

Use `enable` or `disable` to enable/disable, or `help` for this help.

> **Warning:** This does not block sieges, and may not block visitors or monarchs.

6.1.69 hfs-pit

Creates a pit to the underworld at the cursor, taking three numbers as arguments. Usage: `hfs-pit <size> <walls> <stairs>`

The first argument is size of the (square) pit in all directions. The second is `1` to wall off the sides of the pit on all layers except the underworld, or anything else to leave them open. The third parameter is 1 to add stairs. Stairs are buggy; they will not reveal the bottom until you dig somewhere, but underworld creatures will path in.

Examples:

```
hfs-pit 1 0 0
    A single-tile wide pit with no walls or stairs.
    This is the default if no numbers are given.

hfs-pit 4 0 1
    A four-across pit with no stairs but adding walls.

hfs-pit 2 1 0
    A two-across pit with stairs but no walls.
```

6.1.70 hotkey-notes

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

6.1.71 install-info

Saves information about the current DFHack installation to `install-info.txt` in the current DF folder. Useful for bug reports.

6.1.72 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.

6.1.73 launch

Activate with a cursor on screen and you will go there rapidly. Attack something first to ride them there.

6.1.74 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 queue a job at high priority, add `--high` or `--priority`:

```
lever pull 42 --high
```

To magically toggle the lever immediately, add `--now` or `--cheat`:

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

6.1.75 light-aquifers-only

This script behaves differently depending on whether it's called pre embark or post embark. Pre embark it changes all aquifers in the world to light ones, while post embark it only changes the ones at the embark to light ones, leaving the rest of the world unchanged.

Pre embark: Changes the Drainage of all world tiles that would generate Heavy aquifers into a value that results in Light aquifers instead.

This script is based on logic revealed by ToadyOne in a FotF answer: http://www.bay12forums.com/smf/index.php?topic=169696.msg8099138#msg8099138 Basically the Drainage is used as an "RNG" to cause an aquifer to be heavy about 5% of the time. The script shifts the matching numbers to a neighboring one, which does not result in any change of the biome.

Post embark: Clears the flags that mark aquifer tiles as heavy, converting them to light.

6.1.76 linger

Enables the player to take control of their adventurer's killer. Run this script after being presented with "You are deceased."

The killer is identified by examining the historical event generated when the adventurer died. If this is unsuccessful, the killer is assumed to be the last unit to have attacked the adventurer prior to their death.

This will fail if the unit in question is no longer present on the local map.

(Adventure mode only!)

6.1.77 list-agreements

Lists all guildhall and temple agreements in fortress mode.

6.1.78 list-waves

This script displays information about migration waves of the specified citizen(s).

Examples:

```
list-waves -all -showarrival -granularity days
list-waves -all -showarrival
list-waves -unit -granularity days
list-waves -unit
list-waves -unit -all -showarrival -granularity days
```

Selection options:

These options are used to specify what wave information to display

-unit: Displays the highlighted unit's arrival wave information

-all: Displays all citizens' arrival wave information

Other options:

-granularity <value>: Specifies the granularity of wave enumeration: `years`, `seasons`, `months`, `days` If omitted, the default granularity is `seasons`, the same as Dwarf Therapist

-showarrival: Shows the arrival information for the selected unit. If `-all` is specified the info displayed will be relative to the granularity used. Note: this option is always used when `-unit` is used.

6.1.79 load-save

When run on the title screen or "load game" screen, loads the save with the given folder name without requiring interaction. Note that inactive saves (i.e. saves under the "start game" menu) are currently not supported.

Example:

```
load-save region1
```

This can also be run when starting DFHack from the command line. For example, on Linux/macOS:

```
./dfhack +load-save region1
```

6.1.80 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
```

6.1.81 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.

6.1.82 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.

6.1.83 make-monarch

Make the selected unit King or Queen of your civilisation.

6.1.84 markdown

Save a copy of a text screen in markdown (useful for Reddit, among other sites). See *forum-dwarves* for BBCode export (for e.g. the Bay12 Forums).

This script will attempt to read the current df-screen, and if it is a text-viewscreen (such as the dwarf 'thoughts' screen or an item / creature 'description') or an announcement list screen (such as announcements and combat reports) then append a marked-down version of this text to the target file (for easy pasting on reddit for example). Previous entries in the file are not overwritten, so you may use the''markdown'' command multiple times to create a single document containing the text from multiple screens (eg: text screens from several dwarves, or text screens from multiple artifacts/items, or some combination).

Usage:

```
markdown [-n] [filename]
```

-n overwrites contents of output file

filename if provided, save to `md_`*`filename`*`.md` instead of the default `md_export.md`

The screens which have been tested and known to function properly with this script are:

1. dwarf/unit 'thoughts' screen
2. item/art 'description' screen
3. individual 'historical item/figure' screens
4. manual

5. announements screen
6. combat reports screen
7. latest news (when meeting with liaison)

There may be other screens to which the script applies. It should be safe to attempt running the script with any screen active, with an error message to inform you when the selected screen is not appropriate for this script.

6.1.85 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.

6.1.86 migrants-now

Forces an immediate migrant wave. Only works after migrants have arrived naturally. Roughly equivalent to *modtools/force* with:

```
modtools/force -eventType migrants
```

6.1.87 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
```

6.1.88 names

Rename units or items (including artifacts) with the native interface. If a first name is desired, press `f`. To clear the current first name, leave this blank.

6.1.89 on-new-fortress

Runs commands like *multicmd*, but only in a newborn fortress.

Use this in `onMapLoad.init` with f.e. *ban-cooking*:

```
on-new-fortress ban-cooking tallow; ban-cooking honey; ban-cooking oil; ban-cooking␣
↪seeds; ban-cooking brew; ban-cooking fruit; ban-cooking mill; ban-cooking thread;␣
↪ban-cooking milk;
on-new-fortress 3dveins
```

6.1.90 once-per-save

Runs commands like *multicmd*, but only unless not already ran once in current save. You may actually want *on-new-fortress*.

Only successfully ran commands are saved.

Parameters:

--help	display this help
--rerun commands	ignore saved commands
--reset	deletes saved commands

6.1.91 open-legends

Open a legends screen when in fortress mode. Requires a world loaded in fortress or adventure mode. Compatible with *exportlegends*.

Note that this script carries a significant risk of save corruption if the game is saved after exiting legends mode. To avoid this:

1. Pause DF
2. Run *quicksave* to save the game
3. Run *open-legends* (this script) and browse legends mode as usual
4. Immediately after exiting legends mode, run *die* to quit DF without saving (saving at this point instead may corrupt your save)

Note that it should be safe to run "open-legends" itself multiple times in the same DF session, as long as DF is killed immediately after the last run. Unpausing DF or running other commands risks accidentally autosaving the game, which can lead to save corruption.

The optional `force` argument will bypass all safety checks, as well as the save corruption warning.

6.1.92 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.

6.1.93 position

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

6.1.94 pref-adjust

`pref-adjust all` removes/changes preferences from all dwarves, and `pref-adjust one` which works for a single currently selected dwarf. For either, the script inserts an 'ideal' set which is easy to satisfy:

> ... likes iron, steel, weapons, armor, shields/bucklers and plump helmets for their rounded tops. When possible, she prefers to consume dwarven wine, plump helmets, and prepared meals (quarry bush). She absolutely detests trolls, buzzards, vultures and crundles.

Additionally, `pref-adjust goth` will insert a less than ideal set, which is quite challenging, for a single dwarf:

> ... likes dwarf skin, corpses, body parts, remains, coffins, the color black, crosses, glumprongs for their living shadows and snow demons for their horrifying features. When possible, she prefers to consume sewer brew, gutter cruor and bloated tubers. She absolutely detests elves, humans and dwarves.

To see what values can be used with each type of preference, use `pref-adjust list`. Optionally, a single dwarf or all dwarves can have their preferences cleared manually with the use of `pref-adjust clear` and `pref-adjust clear_all`, respectively. Existing preferences are automatically cleared, normally.

6.1.95 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.

6.1.96 prioritize

The prioritize script sets the `do_now` flag on all of the specified types of jobs that are ready to be picked up by a dwarf but not yet assigned to a dwarf. This will force them to get assigned and completed as soon as possible.

This script can also continue to monitor new jobs and automatically boost the priority of jobs of the specified types.

This is useful for ensuring important (but low-priority – according to DF) jobs don't get indefinitely ignored in busy forts. The list of monitored job types is cleared whenever you unload a map, so you can add a section like the one below to your `onMapLoad.init` file to ensure important and time-sensitive job types are always completed promptly in your forts:

```
prioritize -a --haul-labor=Food,Body StoreItemInStockpile
prioritize -a --reaction-name=TAN_A_HIDE CustomReaction
prioritize -a PrepareRawFish ExtractFromRawFish CleanSelf
```

It is important to automatically prioritize only the *most* important job types. If you add too many job types, or if there are simply too many jobs of those types in your fort, the other tasks in your fort can get ignored. This causes the same problem the `prioritize` script is designed to solve. See the *onMapLoad_dreamfort.init* file in the `hack/examples/init` folder for a more complete, playtested set of job types to automatically prioritize.

If you need a bunch of jobs of a specific type prioritized *right now*, consider running `prioritize` without the `-a` parameter, which only affects currently available (but unassigned) jobs. For example:

```
prioritize ConstructBuilding
```

Also see the `do-job-now` *tweak* for adding a hotkey to the jobs screen that can toggle the priority of specific individual jobs and the *do-job-now* script, which boosts the priority of current jobs related to the selected job/unit/item/building/order.

Usage:

```
prioritize [<options>] [<job_type> ...]
```

Examples:

`prioritize` Prints out which job types are being automatically prioritized and how many jobs of each type we have prioritized since we started watching them.

`prioritize -j` Prints out the list of active job types that you can prioritize right now.

`prioritize ConstructBuilding DestroyBuilding` Prioritizes all current building construction and destruction jobs.

`prioritize -a --haul-labor=Food,Body StoreItemInStockpile` Prioritizes all current and future food and corpse hauling jobs.

Options:

`-a`, `--add` Prioritize all current and future jobs of the specified job types.

`-d`, `--delete` Stop automatically prioritizing new jobs of the specified job types.

`-h`, `--help` Show help text.

`-j`, `--jobs` Print out how many unassigned jobs of each type there are. This is useful for discovering the types of the jobs that you can prioritize right now. If any job types are specified, only returns the count for those types.

`-l`, `--haul-labor <labor>[,<labor>...]` For StoreItemInStockpile jobs, match only the specified hauling labor(s). Valid <labor> strings are: "Stone", "Wood", "Body", "Food", "Refuse", "Item", "Furniture", and "Animals". If not specified, defaults to matching all StoreItemInStockpile jobs.

`-n`, `--reaction-name <name>[,<name>...]` For CustomReaction jobs, match only the specified reaction name(s). See the registry output (`-r`) for the full list of reaction names. If not specified, defaults to matching all CustomReaction jobs.

`-q`, `--quiet` Suppress informational output (error messages are still printed).

`-r`, `--registry` Print out the full list of valid job types, hauling labors, and reaction names.

6.1.97 putontable

Makes item appear on the table, like in adventure mode shops. Arguments:

- `-a` or `--all`: apply to all items at the cursor

6.1.98 questport

Teleports your adventurer to the location of your quest log map cursor.

Use by opening the quest log map and moving the cursor to your target location before running the script. Note that this can be done both within and outside of fast travel mode, and that it is possible to questport in situations where fast travel is normally prohibited.

It is currently not possible to questport into inaccessible locations like ocean and mountain tiles.

See *reveal-adv-map* if you wish to teleport into hidden map tiles.

6.1.99 quickfort

Processes Quickfort-style blueprint files.

Quickfort blueprints record what you want at each map coordinate in a spreadsheet, storing the keys in a spreadsheet cell that you would press to make something happen at that spot on the DF map. Quickfort runs in one of five modes: `dig`, `build`, `place`, `zone`, or `query`. `dig` designates tiles for digging, `build` builds buildings and constructions, `place` places stockpiles, `zone` manages activity zones, and `query` changes building or stockpile settings. The mode is determined by a marker in the upper-left cell of the spreadsheet (e.g.: `#dig` in cell `A1`).

You can create these blueprints by hand or by using any spreadsheet application, saving them as `.xlsx` or `.csv` files. You can also build your plan "for real" in Dwarf Fortress, and then export your map using the DFHack *blueprint* plugin (or *gui/blueprint* script) for later replay. Blueprint files should go in the `blueprints` subfolder in the main DF folder.

For more details on blueprint file syntax, see the *Quickfort Blueprint Editing Guide* or browse through the ready-to-use examples in the *Blueprint Library Index*.

Usage:

quickfort set [<key> <value>] Allows you to modify the active quickfort configuration. Just run `quickfort set` to show current settings. See the Configuration section below for available keys and values.

quickfort reset Resets quickfort configuration to the defaults in `quickfort.txt`.

quickfort list [-m|–mode <mode>] [-l|–library] [-h|–hidden] [search string] Lists blueprints in the `blueprints` folder. Blueprints are `.csv` files or sheets within `.xlsx` files that contain a `#<mode>` comment in the upper-left cell. By default, blueprints in the `blueprints/library/` subfolder or blueprints that contain a `hidden()` marker in their modeline are not shown. Specify `-l` or `-h` to include library or hidden blueprints, respectively. The list can be filtered by a specified mode (e.g. "-m build") and/or strings to search for in a path, filename, mode, or comment. The id numbers in the list may not be contiguous if there are hidden or filtered blueprints that are not being shown.

quickfort gui [filename or search terms] Invokes the quickfort UI with the specified parameters, giving you an interactive blueprint preview to work with before you apply it to the map. See the *gui/quickfort* documentation for details.

quickfort <command>[,<command>...] <list_num>[,<list_num>...] [<options>] Applies the blueprint(s) with the number(s) from the `list` command.

quickfort <command>[,<command>...] <filename> [-n|–name <name>[,<name>...]] [<options>] Applies a blueprint in the specified file. The optional `name` parameter can select a specific blueprint from a file that contains multiple blueprints with the format "sheetname/label", or just "/label" for .csv files. The label is defined in the blueprint modeline, or, if not defined, defaults to its order in the sheet or file (e.g. "/2"). If the `-n` parameter is not specified, the first blueprint in the first sheet is used.

<command> is one of:

run Applies the blueprint at your current in-game cursor position.

orders Uses the manager interface to queue up orders to manufacture items for the specified blueprint(s).

undo Applies the inverse of the specified blueprint. Dig tiles are undesignated, buildings are canceled or removed (depending on their construction status), and stockpiles/zones are removed. There is no effect for query blueprints since they can contain arbitrary key sequences.

<options> can be zero or more of:

-c, --cursor <x>,<y>,<z> Use the specified map coordinates instead of the current map cursor for the the blueprint start position. If this option is specified, then an active game map cursor is not necessary.

-d, --dry-run Go through all the motions and print statistics on what would be done, but don't actually change any game state.

--preserve-engravings <quality> Don't designate tiles for digging if they have an engraving with at least the specified quality. Valid values for `quality` are: `None`, `Ordinary`, `WellCrafted`, `FinelyCrafted`, `Superior`, `Exceptional`, and `Masterful`. Specify `None` to ignore engravings when designating tiles. Note that if `Masterful` tiles are dug out, the dwarf who engraved the masterwork will get negative thoughts. If not specified, `Masterful` engravings are preserved by default.

-q, --quiet Suppress non-error console output.

-r, --repeat <direction>[,]<num levels> Repeats the specified blueprint(s) up or down the requested number of z-levels. Direction can be `up` or `down`, and can be abbreviated with < or >. For example, the following options are equivalent: `--repeat down,5`, `-rdown5`, and `-r>5`.

-s, --shift <x>[,<y>] Shifts the blueprint by the specified offset before modifying the game map. The values for `<x>` and `<y>` can be negative. If both `--shift` and `--transform` are specified, the shift is always applied last.

-t, --transform <transformation>[,<transformation>...] Applies geometric transformations to the blueprint before modifying the game map. See the Transformations section below for details.

-v, --verbose Output extra debugging information. This is especially useful if the blueprint isn't being applied like you expect.

Example commands:

```
quickfort list
quickfort list -l dreamfort help
quickfort run library/dreamfort.csv
quickfort run,orders library/dreamfort.csv -n /industry2
quickfort run 10 -dv
```

Transformations:

All transformations are anchored at the blueprint start cursor position. This is the upper left corner by default, but it can be modified if the blueprint has a *start() modeline marker*. This just means that the blueprint tile that would normally appear under your cursor will still appear under your cursor, regardless of how the blueprint is rotated or flipped.

<transformation> is one of:

rotcw or cw Rotates the blueprint 90 degrees clockwise.

rotccw or ccw Rotates the blueprint 90 degrees counterclockwise.

fliph Flips the blueprint horizontally (left edge becomes right edge).

flipv Flips the blueprint vertically (top edge becomes bottom edge).

Configuration:

The quickfort script reads its startup configuration from the `dfhack-config/quickfort/quickfort.txt` file, which you can customize. The settings may be dynamically modified by the `quickfort set` command for the current session, but settings changed with the `quickfort set` command will not change the configuration stored in the file:

`blueprints_dir` (default: 'blueprints') Directory tree to search for blueprints. Can be set to an absolute or relative path. If set to a relative path, resolves to a directory under the DF folder. Note that if you change this directory, you will not see blueprints written by the DFHack *blueprint* plugin (which always writes to the `blueprints` dir) or blueprints in the quickfort blueprint library.

`force_marker_mode` (default: 'false') If true, will designate all dig blueprints in marker mode. If false, only cells with dig codes explicitly prefixed with `m` will be designated in marker mode.

`query_unsafe` (default: 'false') Skip query blueprint sanity checks that detect common blueprint errors and halt or skip keycode playback. Checks include ensuring a configurable building exists at the designated cursor position and verifying the active UI screen is the same before and after sending keys for the cursor position. If you find you need to enable this for one of your own blueprints, you should probably be using a *config blueprint*, not a query blueprint. Most players will never need to enable this setting.

`stockpiles_max_barrels`, `stockpiles_max_bins`, and `stockpiles_max_wheelbarrows` (defaults: -1, -1, 0) Set to the maximum number of resources you want assigned to stockpiles of the relevant types. Set to -1 for DF defaults (number of stockpile tiles for stockpiles that take barrels and bins, 1 wheelbarrow for stone stockpiles). The default here for wheelbarrows is 0 since using wheelbarrows can *decrease* the efficiency of your fort unless you know how to use them properly. Blueprints can *override* this value for specific stockpiles.

There is one other configuration file in the `dfhack-config/quickfort` folder: aliases.txt. It defines keycode shortcuts for query blueprints. The format for this file is described in the *Quickfort Keystroke Alias Guide*, and default aliases that all players can use and build on are available in the *The DFHack standard alias library*. Some quickfort library aliases require the *search* plugin to be enabled.

API:

The quickfort script can be called programmatically by other scripts either via the commandline interface with `dfhack.run_script()` or via the API functions defined in quickfort.lua:

- `apply_blueprint(params)`

Applies the specified blueprint data and returns processing statistics. The statistics structure is a map of stat ids to `{label=string, value=number}`.

`params` is a table with the following fields:

`mode` (required) The name of the blueprint mode, e.g. 'dig', 'build', etc.

`data` (required) A sparse map populated such that `data[z][y][x]` yields the blueprint text that should be applied to the tile at map coordinate `(x, y, z)`. You can also just pass a string and it will be interpreted as the value of `data[0][0][0]`.

`command` The quickfort command to execute, e.g. 'run', 'orders', etc. Defaults to 'run'.

`pos` A coordinate that serves as the reference point for the coordinates in the data map. That is, the text at `data[z][y][x]` will be shifted to be applied to coordinate `(pos.x + x, pos.y + y, pos.z + z)`. If not specified, defaults to `{x=0, y=0, z=0}`, which means that the coordinates in the `data` map are used directly.

`aliases` a map of query blueprint aliases names to their expansions. If not specified, defaults to `{}`.

`preserve_engravings` Don't designate tiles for digging if they have an engraving with at least the specified quality. Value is a df.item_quality enum name or value, or "None" (or, equivalently, -1) to indicate that no engravings should be preserved. Defaults to `df.item_quality.Masterful`.

dry_run Just calculate statistics, such as how many tiles are outside the boundaries of the map; don't actually apply the blueprint. Defaults to false.

verbose Output extra debugging information to the console. Defaults to false.

API usage example:

```
local guidm = require('gui.dwarfmode')
local quickfort = reqscript('quickfort')
-- dig a 10x10 block at the cursor position
quickfort.apply_blueprint{mode='dig', data={[0]={[0]={[0]='d(10x10)'}}},
                          pos=guidm.getCursorPos()}
```

6.1.100 quicksave

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

Keybinding: CtrlAltS in `dwarfmode/Default`

6.1.101 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.

6.1.102 rejuvenate

Decreases the age of the selected dwarf to 20 years. Useful if valuable citizens are getting old.

Arguments:

- `-all`: applies to all citizens
- `-force`: also applies to units under 20 years old. Useful if there are too many babies around...
- `-dry-run`: only list units that would be changed; don't actually change ages

6.1.103 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.

Using the argument `-value 0` will reduce stress to the value 0 instead of -1,000,000. Negative values must be preceded by a backslash (): `-value \-10000`. Note that this can only be used to *decrease* stress - it cannot be increased with this argument.

6.1.104 remove-wear

Sets the wear on items in your fort to zero. Usage:

remove-wear all Removes wear from all items in your fort.

remove-wear ID1 ID2 ... Removes wear from items with the given ID numbers.

6.1.105 repeat

Repeatedly calls a lua script at the specified interval. This can be used from init files. Note that any time units other than `frames` are unsupported when a world is not loaded (see `dfhack.timeout()`).

Usage examples:

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

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

Arguments:

`-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` (default "ticks"). Units can be in simulation-time "frames" (raw FPS) or "ticks" (only while unpaused), while "days", "months", and "years" are by in-world time.

`-command [ ... ] ...` specifies the command to be run

`-cancel NAME` cancels the repetition with the name NAME

`-list` prints names of scheduled commands

6.1.106 resurrect-adv

Brings a dead adventurer back to life, fully healing them in the process.

This script only targets the current player character in your party, and should be run after being presented with the "You are deceased" message. It is not possible to resurrect the adventurer after the game has been ended.

6.1.107 reveal-adv-map

This script can be used to either reveal or hide all tiles on the world map in adventure mode (visible when viewing the quest log or fast travelling, for example).

Note that the script does not reveal hidden lairs, camps, etc. See *reveal-hidden-sites* for this functionality.

Arguments:

```
-hide
    Include this if you want to hide all world tiles instead
    of revealing them
```

6.1.108 reveal-hidden-sites

This script reveals all sites in the world that have yet to be discovered by the player (camps, lairs, shrines, vaults, etc) thus making them visible on the map.

Usable in both fortress and adventure mode.

See *reveal-adv-map* if you also want to expose hidden world map tiles in adventure mode.

6.1.109 reveal-hidden-units

This script exposes all units on the map who are currently sneaking or waiting in ambush and thus hidden from the player's view.

Usable in both fortress and adventure mode.

6.1.110 season-palette

Swap color palettes when the seasons change.

For this script to work you need to add *at least* one color palette file to your save raw directory.

Palette file names: "colors.txt": The world "default" (worldgen and replacement) palette. "colors_spring.txt": The palette displayed during spring. "colors_summer.txt": The palette displayed during summer. "colors_autumn.txt": The palette displayed during autumn. "colors_winter.txt": The palette displayed during winter.

If you do not provide a world default palette, palette switching will be disabled for the current world. The seasonal palettes are optional, the default palette is not! The default palette will be used to replace any missing seasonal palettes and during worldgen.

When the world is unloaded or this script is disabled, the system default color palette ("/data/init/colors.txt") will be loaded. The system default palette will always be used in the main menu, but your custom palettes should be used everywhere else.

Usage:

`season-palette start|enable` or `enable season-palette`: Begin swapping seasonal color palettes.

`season-palette stop|disable` or `disable season-palette`: Stop swapping seasonal color palettes and load the default color palette.

If loaded as a module this script will export a single Lua function:

`LoadPalette(path)`: Load a color palette from the text file at "path". This file must be in the same format as "/data/init/colors.txt". If there is an error any changes will be reverted and this function will return false (returns true normally).

6.1.111 set-orientation

Edit a unit's orientation. Interest levels are 0 for Uninterested, 1 for Romance, 2 for Marry.

unit <UNIT ID> The given unit will be affected. If not found/provided, the script will try defaulting to the currently selected unit.

male <INTEREST> Set the interest level towards male sexes

female <INTEREST> Set the interest level towards female sexes

opposite <INTEREST> Set the interest level towards the opposite sex to the unit

same <INTEREST> Set the interest level towards the same sex as the unit

random Randomise the unit's interest towards both sexes, respecting their ORIENTATION token odds.

Other arguments:

help Shows this help page.

view Print the unit's orientation values in the console.

6.1.112 set-timeskip-duration

Starting a new fortress/adventurer session is preceded by an "Updating World" process which is normally 2 weeks long. This script allows you to modify the duration of this timeskip, enabling you to jump into the game earlier or later than usual.

You can use this at any point before the timeskip begins (for example, while still at the "Start Playing" menu).

It is also possible to run the script while the world is updating, which can be useful if you decide to end the process earlier or later than initially planned.

Note that the change in timeskip duration will persist until either

- the game is closed,
- the "-clear" argument is used (see below),
- or it is overwritten by setting a new duration.

The timeskip duration can be specified using any combination of the following arguments.

Usage:

```
-ticks X
    Replace "X" with a positive integer (or 0)
    Adds the specified number of ticks to the timeskip duration
    The following conversions may help you calculate this:
        1 tick = 72 seconds = 1 minute 12 seconds
        50 ticks = 60 minutes = 1 hour
        1200 ticks = 24 hours = 1 day
        8400 ticks = 7 days = 1 week
        33600 ticks = 4 weeks = 1 month
        403200 ticks = 12 months = 1 year

-years X
    Replace "X" with a positive integer (or 0)
    Adds (403200 multiplied by X) ticks to the timeskip duration

-months X
```

(continues on next page)

(continued from previous page)

```
    Replace "X" with a positive integer (or 0)
    Adds (33600 multiplied by X) ticks to the timeskip duration

-days X
    Replace "X" with a positive integer (or 0)
    Adds (1200 multiplied by X) ticks to the timeskip duration

-hours X
    Replace "X" with a positive integer (or 0)
    Adds (50 multiplied by X) ticks to the timeskip duration

-clear
    This resets the timeskip duration to its default value.
    Note that it won't affect timeskips which have already begun.
```

Example:

```
set-timeskip-duration -ticks 851249
    Sets the end of the timeskip to
    2 years, 1 month, 9 days, 8 hours, 58 minutes, 48 seconds
    from the current date.

set-timeskip-duration -years 2 -months 1 -days 9 -hours 8 -ticks 49
    Does the same thing as the previous example
```

6.1.113 setfps

Sets the FPS cap at runtime. Useful in case you want to speed up the game or watch combat in slow motion.

Usage:

```
setfps <number>
```

6.1.114 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.

6.1.115 siren

Wakes up sleeping units and stops parties, either everywhere or in the burrows given as arguments. In return, adds bad thoughts about noise, tiredness and lack of protection. The script is intended for emergencies, e.g. when a siege appears, and all your military is partying.

6.1.116 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 every time, 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
```

6.1.117 spawnunit

Provides a simpler interface to *modtools/create-unit*, for creating units.

Usage: `spawnunit [-command] RACE CASTE [NAME] [x y z] [...]`

The `-command` flag prints the generated *modtools/create-unit* command instead of running it. `RACE` and `CASTE` specify the race and caste of the unit to be created. The name and coordinates of the unit are optional. Any further arguments are simply passed on to *modtools/create-unit*.

6.1.118 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.

6.1.119 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)`

6.1.120 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
```

6.1.121 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' periods of targets.

6.1.122 tame

Tame and train animals.

Usage: `tame -read` OR `tame -set <level>`

0 semi-wild

1 trained

2 well-trained

3 skillfully trained

4 expertly trained

5 exceptionally trained

6 masterfully trained

7 tame

8 wild

9 wild

6.1.123 teleport

Teleports a unit to given coordinates.

Note: *gui/teleport* is an in-game UI for this script.

Examples:

- prints ID of unit beneath cursor:

```
teleport -showunitid
```

- prints coordinates beneath cursor:

```
teleport -showpos
```

- teleports unit `1234` to `56,115,26`

  teleport -unit 1234 -x 56 -y 115 -z 26

6.1.124 tidlers

Toggle between all possible positions where the idlers count can be placed (see `data/init/d_init.txt` for details).

6.1.125 timestream

Controls the speed of the calendar and creatures. Fortress mode only. Experimental.

The script is also capable of dynamically speeding up the game based on your current FPS to mitigate the effects of FPS death. See examples below to see how.

Usage:

```
timestream [-rate R] [-fps FPS] [-units [FLAG]] [-debug]
```

Examples:

- **`timestream -rate 2`:** Calendar runs at x2 normal speed, units run at normal speed
- **`timestream -fps 100`:** Calendar runs at dynamic speed to simulate 100 FPS, units normal
- **`timestream -fps 100 -units`:** Calendar & units are simulated at 100 FPS
- **`timestream -rate 1`:** Resets everything back to normal, regardless of other arguments

- **`timestream -rate 1 -fps 50 -units`:** Same as above
- **`timestream -fps 100 -units 2`:** Activates a different mode for speeding up units, using the native DF `debug_turbospeed` flag (similar to *fastdwarf* 2) instead of adjusting timers of all units. This results in rubberbanding unit motion, so it is not recommended over the default method.

Original timestream.lua: https://gist.github.com/IndigoFenix/cf358b8c994caa0f93d5

6.1.126 troubleshoot-item

Print various properties of the selected item. Sometimes useful for troubleshooting issues such as why dwarves won't pick up a certain item.

6.1.127 twaterlvl

Toggle between displaying/not displaying liquid depth as numbers.

Keybinding: CtrlW

6.1.128 undump-buildings

Undesignates building base materials for dumping.

6.1.129 unforbid

Unforbids all items.

Usage:

```
unforbid all
unforbid help
```

6.1.130 ungeld

A wrapper around *geld* that ungelds the specified animal.

Valid options:

`-unit <id>`: Ungelds the unit with the specified ID. This is optional; if not specified, the selected unit is used instead.

`-help`: Shows this help information

6.1.131 uniform-unstick

Prompt units to reevaluate their uniform, by removing/dropping potentially conflicting worn items.

Unlike a "replace clothing" designation, it won't remove additional clothing if it's coexisting with a uniform item already on that body part. It also won't remove clothing (e.g. shoes, trousers) if the unit has yet to claim an armor item for that bodypart. (e.g. if you're still manufacturing them.)

By default it simply prints info about the currently selected unit, to actually drop items, you need to provide it the -drop option.

The default algorithm assumes that there's only one armor item assigned per body part, which means that it may miss cases where one piece of armor is blocked but the other is present. The -multi option can possibly get around this, but at the cost of ignoring left/right distinctions when dropping items.

In some cases, an assigned armor item can't be put on because someone else is wearing/holding it. The -free option will cause the assigned item to be removed from the container/dwarven inventory and placed onto the ground, ready for pickup.

In no cases should the command cause a uniform item that is being properly worn to be removed/dropped.

Targets:

(no target) Force the selected dwarf to put on their uniform.

-all Force the uniform on all military dwarves.

Options:

(none) Simply show identified issues (dry-run).

-drop Cause offending worn items to be placed on ground under unit.

-free Remove to-equip items from containers or other's inventories, and place on ground.

-multi Be more agressive in removing items, best for when uniforms have muliple items per body part.

6.1.132 unretire-anyone

This script allows the user to play as any living (or undead) historical figure (except for deities) in adventure mode.

To use, simply enter "unretire-anyone" in the DFHack console at the start of adventure mode character creation. You will be presented with a searchable list from which you may choose your desired historical figure.

This figure will be added to the 'Specific Person' list at the bottom of the creature selection page. They can then be picked for use as a player character, as if regaining control of a retired adventurer.

6.1.133 unsuspend

Unsuspends building construction jobs, except for jobs managed by *buildingplan* and those where water flow > 1. See *autounsuspend* for keeping new jobs unsuspended.

6.1.134 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* 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.

6.1.135 view-unit-reports

Show combat reports for the current unit.

Current unit is unit near cursor in 'v', selected in 'u' list, unit/corpse/splatter at cursor in 'k'. And newest unit with race when 'k' at race-specific blood spatter.

Keybinding: CtrlShiftR

6.1.136 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. If you only want to be warned about sane dwarves, use `warn-starving sane`.

Use with the *repeat* command for regular checks.

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

6.1.137 weather

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

6.1.138 workorder

`workorder` is a script to queue work orders as in `j-m-q` menu. It can automatically count how many creatures can be milked or sheared.

The most simple and obvious usage is automating shearing and milking of creatures using `repeat`:

```
repeat -time 14 -timeUnits days -command [ workorder ShearCreature ] -name␣
↪autoShearCreature
repeat -time 14 -timeUnits days -command [ workorder MilkCreature ] -name␣
↪autoMilkCreature
```

It is also possible to define complete work orders using `json`. It is very similar to what `orders import filename` does, with a few key differences. `workorder` is a planning tool aiming to provide scripting support for vanilla manager. As such it will ignore work order state like `amount_left` or `is_active` and can optionally take current orders into account. See description of `<json>`-parameter for more details.

Examples:

- `workorder ShearCreature 10` add an order to "Shear Animal" 10 times.
- `workorder ShearCreature` same, but calculate amount automatically (can be 0).
- `workorder MilkCreature` same, but "Milk Animal".

Advanced examples:

- **`workorder "{\"job\":\"EncrustWithGems\",\"item_category\":[\"finished_goods\"],\"amoun`**
  add an order to `EncrustWithGems finished_goods` using any material (since not specified).
- **`workorder "{\"job\":\"MilkCreature\",\"item_conditions\":[{\"condition\":\"AtLeast\",\`**
  same as `workorder MilkCreature` but with an item condition ("at least 2 empty buckets").

Usage:

```
workorder [ --<command> | <jobtype> [<amount>] | <json> | --file <file> ]
```

<command> one of `help`, `listtypes`, `verbose`, `very-verbose`

--help	this help.
--listtypes filter	print all values for all used DF types (`job_type`, `item_type` etc.). `<filter>` is optional and is applied to type name (using `Lua`'s `string.find`), f.e. `workorder -l "manager"` is useful.

<jobtype> number or name from `df.job_type`.

<amount> optional number; if omitted, the script will try to determine amount automatically for some jobs. Currently supported are `MilkCreature` and `ShearCreature` jobs.

<json> json-representation of a workorder. Must be a valid Lua string literal (see advanced examples: note usage of `\`). Use `orders export some_file_name` to get an idea how does the `json`-structure look like.

It's important to note this script behaves differently compared to `orders import some_file_name`: `workorder` is meant as a planning tool and as such it **will ignore** some fields like `amount_left`, `is_active` or `is_validated`.

This script doesn't need values in all fields:

- `id` is only used for order conditions;
- `frequency` is set to `OneTime` by default;
- `amount_total` can be missing, a function name from this script (one of `calcAmountFor_MilkCreature` or `calcAmountFor_ShearCreature`) or `Lua` code called as `load(code)(order, orders)`. Missing `amount_total` is equivalent to `calcAmountFor_<order.job>`.

A custom field `__reduce_amount` can be set if existing open orders should be taken into account reducing new order's `total_amount` (possibly all the way to `0`). An empty `amount_total` implies `"__reduce_amount": true`.

--file filename loads the json-representation of a workorder from a file in `dfhack-config/workorder/`.

Debugging:

--verbose toggle script's verbosity.

--very-verbose toggle script's very verbose mode.

--reset reset script environment for next execution.

6.1.139 workorder-recheck

Sets the status to `Checking` (from `Active`) of the selected work order (in the `j-m` or `u-m` screens). This makes the manager reevaluate its conditions.

Keybinding: AltR in `jobmanagement/Main`

6.2 Development Scripts

`devel/*` scripts are intended for developer use, but many may be of interest to anyone investigating odd phenomena 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/block-borders*
- *devel/check-other-ids*
- *devel/check-release*
- *devel/clear-script-env*
- *devel/click-monitor*
- *devel/cmptiles*
- *devel/dump-offsets*
- *devel/eventful-client*
- *devel/export-dt-ini*
- *devel/find-offsets*
- *devel/find-primitive*
- *devel/find-twbt*
- *devel/inject-raws*
- *devel/inspect-screen*
- *devel/kill-hf*
- *devel/light*
- *devel/list-filters*
- *devel/lsmem*
- *devel/lua-example*
- *devel/luacov*
- *devel/modstate-monitor*
- *devel/nuke-items*
- *devel/pop-screen*
- *devel/prepare-save*
- *devel/print-args*
- *devel/print-args2*
- *devel/print-event*
- *devel/query*
- *devel/save-version*
- *devel/sc*
- *devel/scanitemother*
- *devel/send-key*
- *devel/spawn-unit-helper*
- *devel/test-perlin*
- *devel/unforbidall*

- *devel/unit-path*
- *devel/visualize-structure*
- *devel/watch-minecarts*

6.2.1 devel/all-bob

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

6.2.2 devel/annc-monitor

Displays announcements and reports in the console.

enable|start Begins monitoring

disable|stop Stops monitoring

report enable Show combat reports

report disable Only show announcements

6.2.3 devel/block-borders

An overlay that draws borders of map blocks. See *Maps API* for details on map blocks.

6.2.4 devel/check-other-ids

This script runs through all world.items.other and world.buildings.other vectors and verifies that the items contained in them have the expected types.

6.2.5 devel/check-release

Basic checks for release readiness

6.2.6 devel/clear-script-env

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

6.2.7 devel/click-monitor

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

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

6.2.8 devel/cmptiles

Lists and/or compares two tiletype material groups.

Usage: `devel/cmptiles material1 [material2]`

6.2.9 devel/dump-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.

This dumps the contents of the table of global addresses (new in 0.44.01).

Passing global names as arguments calls setAddress() to set those globals' addresses in-game. Passing "all" does this for all globals.

6.2.10 devel/eventful-client

Usage:

```
devel/eventful-client help
devel/eventful-client add <event type> <frequency>
devel/eventful-client add all <frequency>
devel/eventful-client list
devel/eventful-client clear
```

help shows this help text and a list of valid event types

add add a handler for the named event type at the requested tick frequency

list lists active handlers and their metadata

clear unregisters all handlers

Note this script does not handle the eventful reaction or workshop events.

6.2.11 devel/export-dt-ini

Exports an ini file containing memory addresses for Dwarf Therapist.

6.2.12 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

6.2.13 devel/find-primitive

Finds a primitive variable in DF's data section, relying on the user to change its value. This is similar to *devel/find-offsets*, but useful for new variables whose locations are unknown (i.e. they could be part of an existing global).

Usage:

```
devel/find-primitive data-type val1 val2 [val3...]
```

where `data-type` is a primitive type (int32_t, uint8_t, long, etc.) and each `val` is a valid value for that type.

Use `devel/find-primitive help` for a list of valid data types.

6.2.14 devel/find-twbt

Finds some TWBT-related offsets - currently just `twbt_render_map`.

6.2.15 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 reaction STOKE_BOILER
```

6.2.16 devel/inspect-screen

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

6.2.17 devel/kill-hf

Kills the specified historical figure, even if off-site, or terminates a pregnancy. Useful for working around Bug 11549.

Usage:

```
devel/kill-hf [-p|--pregnancy] [-n|--dry-run] HISTFIG_ID
```

Arguments:

histfig_id: the ID of the historical figure to target

-p, --pregnancy: if specified, and if the historical figure is pregnant, terminate the pregnancy instead of killing the historical figure

-n, --dry-run: if specified, only print the name of the historical figure

6.2.18 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.

6.2.19 devel/list-filters

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

6.2.20 devel/lsmem

Prints memory ranges of the DF process. Useful for checking whether a pointer is valid, whether a certain library/plugin is loaded, etc.

6.2.21 devel/lua-example

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

6.2.22 devel/luacov

Lua script coverage report generator

Usage:

    luacov [options] [pattern...]

This script generates a coverage report from collected statistics. By default it reports on every Lua file in all of DFHack. To filter filenames, specify one or more Lua patterns matching files or directories to be included. Alternately, you can configure reporting parameters in the .luacov file in your DF directory. See https://keplerproject.github.io/luacov/doc/modules/luacov.defaults.html for details.

Statistics are cumulative across reports. That is, if you run a report, run a lua script, and then run another report, the report will include all activity from the first report plus the recently run lua script. Restarting DFHack will clear the statistics. You can also clear statistics after running a report by passing the –clear flag to this script.

Note that the coverage report will be empty unless you have started DFHack with the "DFHACK_ENABLE_LUACOV=1" environment variable defined, which enables the coverage monitoring.

Also note that enabling both coverage monitoring and lua profiling via the "profiler" module can produce strange results. Their interceptor hooks override each other. Usage of the "kill-lua" command will likewise override the luacov interceptor hook and may prevent coverage statistics from being collected.

Options:

-c, --clear Remove accumulated metrics after generating the report, ensuring the next report starts from a clean slate.

-h, --help Show this help message and exit.

Examples:

devel/luacov Report on all DFHack lua scripts.

devel/luacov -c quickfort Report only on quickfort source files and clear stats.

devel/luacov quickfort hack/lua Report only on quickfort and DFHack library lua source files.

6.2.23 devel/modstate-monitor

Display changes in key modifier state, i.e. `Ctrl`/`Alt`/`Shift`.

Arguments:

enable|start Begin monitoring

disable|stop End monitoring

6.2.24 devel/nuke-items

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

6.2.25 devel/pop-screen

Forcibly closes the current screen. This is usually equivalent to pressing `Esc` (`LEAVESCREEN`), but will bypass the screen's input handling. This is intended primarily for development, if you have created a screen whose input handling throws an error before it handles `Esc` (or if you have forgotten to handle `Esc` entirely).

Warning: If you run this script when the current screen does not have a parent, this will cause DF to exit **immediately**. These screens include:

- The main fortress mode screen (`viewscreen_dwarfmodest`)
- The main adventure mode screen (`viewscreen_dungeonmodest`)
- The main legends mode screen (`viewscreen_legendsst`)
- The title screen (`viewscreen_titlest`)

6.2.26 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.

6.2.27 devel/print-args

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

6.2.28 devel/print-args2

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

6.2.29 devel/print-event

Prints the description of an event by ID or index.

6.2.30 devel/query

Query is a script useful for finding and reading values of data structure fields. Purposes will likely be exclusive to writing lua script code, possibly C++.

This script takes your data selection eg.{table,unit,item,tile,etc.} then recursively iterates through it, outputting names and values of what it finds.

As it iterates you can have it do other things, like search for a specific structure pattern (see lua patterns) or set the value of fields matching the selection and any search pattern specified.

Note: This is a recursive search function. The data structures are also recursive. So there are a few things that must be considered (in order):

- Is the search depth too high? (Default: 7)
- Is the data capable of being iterated, or does it only have a value?
- How can the data be iterated?
- Is the iteration count for the data too high? (Default: 257)
- Does the user want to exclude the data's type?
- Is the data recursively indexing (eg. A.B.C.A.*)?
- Does the data match the search pattern?

Warning: This is a recursive script that's primary use is to search recursive data structures. You can, fairly easily, cause an infinite loop. You can even more easily run a query that simply requires an inordinate amount of time to complete.

Tip: Should the need arise, you can kill the command from another shell with *kill-lua*, e.g. by running it with *dfhack-run* from another terminal.

Usage examples:

```
devel/query -unit -getfield id
devel/query -unit -search STRENGTH
devel/query -unit -search physical_attrs -maxdepth 2
devel/query -tile -search dig
devel/query -tile -search "occup.*carv"
devel/query -table df -maxdepth 2
devel/query -table df -maxdepth 2 -excludekinds s -excludetypes fsu -oneline
devel/query -table df.profession -findvalue FISH
devel/query -table df.global.ui.main -maxdepth 0
devel/query -table df.global.ui.main -maxdepth 0 -oneline
devel/query -table df.global.ui.main -maxdepth 0 -1
```

Selection options:

-tile Selects the highlighted tile's block, and then uses the tile's local position to index the 2D data.

-block Selects the highlighted tile's block.

-unit Selects the highlighted unit

-item Selects the highlighted item.

-plant Selects the highlighted plant.

-building Selects the highlighted building.

-job Selects the highlighted job.

-script <script name> Selects the specified script (which must support being included with `reqscript()`).

-json <file> Loads the specified json file as a table to query.

> **Note:** The path starts at the DF root directory. eg. -json /hack/scripts/dwarf_profiles.json

-table <identifier> Selects the specified table (ie. 'value').

> **Note:** You must use dot notation to denote sub-tables. eg. `df.global.world`

-getfield <name> Gets the specified field from the selection.

Must use in conjunction with one of the above selection options. Must use dot notation to denote sub-fields.

Query options:

-search <pattern> Searches the selection for field names with substrings matching the specified value.

Usage:

```
devel/query -table dfhack -search pattern
devel/query -table dfhack -search [ pattern1 pattern2 ]
```

-findvalue <value> Searches the selection for field values matching the specified value.

-maxdepth <value> Limits the field recursion depth (default: 7)

-maxlength <value> Limits the table sizes that will be walked (default: 257)

-excludetypes [a|bfnstu0] Excludes native Lua data types. Single letters correspond to (in order): All types listed here, Boolean, Function, Number, String, Table, Userdata, nil

-excludekinds [a|bces] Excludes DF data types. Single letters correspond to (in order): All types listed here, Bitfield-type, Class-type, Enum-type, Struct-type

-dumb Disables intelligent checking for recursive data structures (loops) and increases the `-maxdepth` to 25 if a value is not already present

General options:

-showpaths Displays the full path of a field instead of indenting.

-setvalue <value> Attempts to set the values of any printed fields. Supported types: boolean, string, integer

-oneline, -1 Reduces output to one line, except with `-debugdata`

-alignto <value> Specifies the alignment column.

-nopointers Disables printing values which contain memory addresses.

-disableprint Disables printing. Might be useful if you are debugging this script. Or to see if a query will crash (faster) but not sure what else you could use it for.

-debug <value> Enables debug log lines equal to or less than the value provided.

-debugdata Enables debugging data. Prints type information under each field.

-help Prints this help information.

6.2.31 devel/save-version

Display DF version information about the current save

6.2.32 devel/sc

Size Check: scans structures for invalid vectors, misaligned structures, and unidentified enum values.

Note: This script can take a very long time to complete, and DF may be unresponsive while it is running. You can use *kill-lua* to interrupt this script.

Examples:

- scan world:

```
devel/sc
```

- scan all globals:

```
devel/sc -all
```

- scan result of expression:

```
devel/sc [expr]
```

6.2.33 devel/scanitemother

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

6.2.34 devel/send-key

Send a key to the current screen or a parent. If you are trying to dismiss a screen, *devel/pop-screen* may be more useful, particularly if the screen is unresponsive to `Esc`.

Usage:

```
devel/send-key KEY_NAME [X]
```

- `KEY_NAME` is the name of an `interface_key` - a full list of valid names can be obtained by running `lua @df.interface_key`, looking in `data/init/interface.txt`, or by checking `df.keybindings.xml` in the df-structures repository.
- `X` (optional) - if specified, the key is sent to the screen `X` screens above the current screen in the screen stack (e.g. `1` corresponds to the current screen's parent)

6.2.35 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`

6.2.36 devel/test-perlin

Generates an image using multiple octaves of perlin noise.

6.2.37 devel/unforbidall

Unforbid all items.

6.2.38 devel/unit-path

Show the internal path a unit is currently following.

6.2.39 devel/visualize-structure

Displays the raw memory of a structure, separated by field. Useful for checking if structures are aligned.

6.2.40 devel/watch-minecarts

Logs minecart coordinates and speeds to console.

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

6.3 Bugfixing Scripts

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

Contents

6.3.1 fix/blood-del

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

6.3.2 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.

6.3.3 fix/corrupt-equipment

Fixes some corruption that can occur in equipment lists, as in Bug 11014.

Note that there have been several possible types of corruption identified:

1. Items that have been deleted without being removed from the equipment lists

2. Items of the wrong type being stored in the equipment lists
3. Items of the wrong type being assigned to squad members

This script currently only fixes the first two, as they have been linked to the majority of crashes.

Note that in some cases, multiple issues may be present, and may only be present for a short window of time before DF crashes. To address this, running this script with *repeat* is recommended. For example, to run this script every 100 ticks:

```
repeat -name fix-corrupt-equipment -time 100 -timeUnits ticks -command [ fix/corrupt-
↪equipment ]
```

To cancel it (which is likely safe if the script has not produced any output in some time, and if you have saved first):

```
repeat -cancel fix-corrupt-equipment
```

Running this script with *repeat* on all saves is not recommended, as it can have overhead (sometimes over 0.1 seconds on a large save). In general, running this script with *repeat* is recommended if:

- You have experienced crashes likely caused by Bug 11014, and running this script a single time produces output but does not fix the crash
- You are running large military operations, or have sent squads out on raids

6.3.4 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.

6.3.5 fix/diplomats

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 version 0.31.12, in which the "bug" was "fixed".

6.3.6 fix/drop-webs

Turns floating webs into projectiles, causing them to fall down to a valid surface. This addresses Bug 595.

Use `fix/drop-webs -all` to turn all webs into projectiles, causing webs to fall out of branches, etc.

Use *clear-webs* to remove webs entirely.

6.3.7 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.

6.3.8 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.

6.3.9 fix/feeding-timers

Reset the GiveWater and GiveFood timers of all living citizens.

6.3.10 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

6.3.11 fix/loyaltycascade

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

6.3.12 fix/merchants

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

6.3.13 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.

6.3.14 fix/retrieve-units

This script forces some units off the map to enter the map, which can fix issues such as the following:

- Stuck [SIEGE] tags due to invisible armies (or parts of armies)
- Forgotten beasts that never appear
- Packs of wildlife that are missing from the surface or caverns
- Caravans that are partially or completely missing.

Note: For caravans that are missing entirely, this script may retrieve the merchants but not the items. Using *fix/stuck-merchants* followed by *force* to create a new caravan may work better.

6.3.15 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*.

6.3.16 fix/stuck-merchants

Dismisses merchants that haven't entered the map yet. This can fix Bug 9593. This script should probably not be run if any merchants are on the map, so using it with *repeat* is not recommended.

Run `fix/stuck-merchants -n` or `fix/stuck-merchants --dry-run` to list all merchants that would be dismissed but make no changes.

6.3.17 fix/stuckdoors

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

6.3.18 fix/tile-occupancy

Clears bad occupancy flags at the selected tile. Useful for getting rid of phantom "building present" messages. Currently only supports issues with building and unit occupancy. Requires that a tile is selected with the in-game cursor (`k`).

Can be used to fix problematic tiles caused by Issue 1047.

6.4 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/dfstatus*
- *gui/extended-status*
- *gui/family-affairs*
- *gui/gm-editor*
- *gui/gm-unit*
- *gui/guide-path*
- *gui/hack-wish*
- *gui/hello-world*
- *gui/liquids*
- *gui/load-screen*
- *gui/manager-quantity*
- *gui/mass-remove*
- *gui/mechanisms*
- *gui/mod-manager*
- *gui/no-dfhack-init*
- *gui/pathable*
- *gui/power-meter*
- *gui/prerelease-warning*
- *gui/quickcmd*
- *gui/quickfort*
- *gui/rename*
- *gui/room-list*
- *gui/settings-manager*
- *gui/siege-engine*
- *gui/stamper*
- *gui/stockpiles*
- *gui/teleport*
- *gui/unit-info-viewer*
- *gui/workflow*
- *gui/workshop-job*

6.4.1 gui/advfort

This script allows performing jobs in adventure mode. For more complete help press ? while the script is running. It's most comfortable to use this as a keybinding (see below for the default binding). Possible arguments:

- **`-a`, `--nodfassign`:** uses a different method to assign job items, instead of relying on DF.

- **-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
- **-e [NAME], --entity [NAME]:** uses the given civ to determine available resources (specified as an entity raw ID). Defaults to `MOUNTAIN`; if the entity name is omitted, uses the adventurer's civ
- `job`: selects the specified job (must be a valid `job_type`, e.g. `Dig` or `FellTree`)

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

An example of a player digging in adventure mode:

Keybinding: CtrlT in `dungeonmode`

6.4.2 gui/advfort_items

A module to support item handling in *gui/advfort* - not intended to be called directly.

6.4.3 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.

6.4.4 gui/autobutcher

An in-game interface for *autobutcher*. This script must be called from either the overall status screen or the animal list screen.

Keybinding: ShiftB in `pet/List/Unit`

6.4.5 gui/autogems

A frontend for the *autogems* plugin that allows configuring the gem types to be cut.

The following controls apply to the gems currently listed:

- `s`: Searches for matching gems
- `Shift+Enter`: Toggles the status of all listed gems

The following controls apply to the gems currently listed, as well as gems listed *before* the current search with `s`, if applicable:

- `r`: Displays only "rock crystal" gems
- `c`: Displays only gems whose color matches the selected gem
- `m`: Displays only gems where at least one rough (uncut) gem is available somewhere on the map

This behavior is intended to allow for things like a search for "lazuli" followed by pressing `c` to select all gems with the same color as lapis lazuli (5 blue gems in vanilla DF), rather than further restricting that to gems with "lazuli" in their name (only 1).

`x` clears all filters, which is currently the only way to undo filters (besides searching), and is useful to verify the gems selected.

6.4.6 gui/blueprint

The *blueprint* plugin records the structure of a portion of your fortress in a blueprint file that you (or anyone else) can later play back with *quickfort*.

This script provides a visual, interactive interface to make configuring and using the blueprint plugin much easier.

Usage:

```
gui/blueprint [<name> [<phases>]] [<options>]
```

All parameters are optional. Anything you specify will override the initial values set in the interface. See the *blueprint* documentation for information on the possible parameters and options.

6.4.7 gui/choose-weapons

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.

Keybinding: CtrlW in `layer_military/Equip/Customize/View`

6.4.8 gui/clone-uniform

When invoked, the script duplicates the currently selected uniform template, and selects the newly created copy. Activate in the Uniforms page of the military screen with the cursor in the leftmost list.

Keybinding: CtrlC in `layer_military/Uniforms`

6.4.9 gui/color-schemes

An in-game interface for *color-schemes*. This script must be called from either the title screen or the dwarf fortress default screen.

6.4.10 gui/companion-order

A script to issue orders for companions. Select companions with lower case chars (green when selected), issue orders with upper case. Must be in look or talk mode to issue command on tile (e.g. move/equip/pick-up).

- 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)

Can be called with '-c' flag to display "cheating" commands.

- patch up - fully heals the companion
- get in - rides e.g. minecart at cursor. Bit buggy as unit will teleport to the item when e.g. pushing it.

Keybinding: ShiftO in `dungeonmode`

6.4.11 gui/confirm-opts

A basic configuration interface for the *confirm* plugin.

6.4.12 gui/cp437-table

An in-game CP437 table. Allows typing or selecting characters with the mouse. Input is fed directly to the parent screen when "enter" is pressed, so there should be a text field selected before running this script.

6.4.13 gui/create-item

A graphical interface for creating items.

See also: *createitem*, *modtools/create-item*, Issue 735

6.4.14 gui/create-tree

A graphical interface for creating trees.

Place the cursor wherever you want the tree to appear and run the script. Then select the desired tree type from the list. You will then be asked to input the desired age of the tree in years. If omitted, the age will default to 1.

6.4.15 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`.

Keybinding: CtrlShiftI in `dwarfmode/Default`

Keybinding: CtrlShiftI in `dfhack/lua/dfstatus`

6.4.16 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
```

6.4.17 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 specified units and their partners, then arranges for the two units to marry, bypassing almost all checks. Use with caution.

6.4.18 gui/gm-editor

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

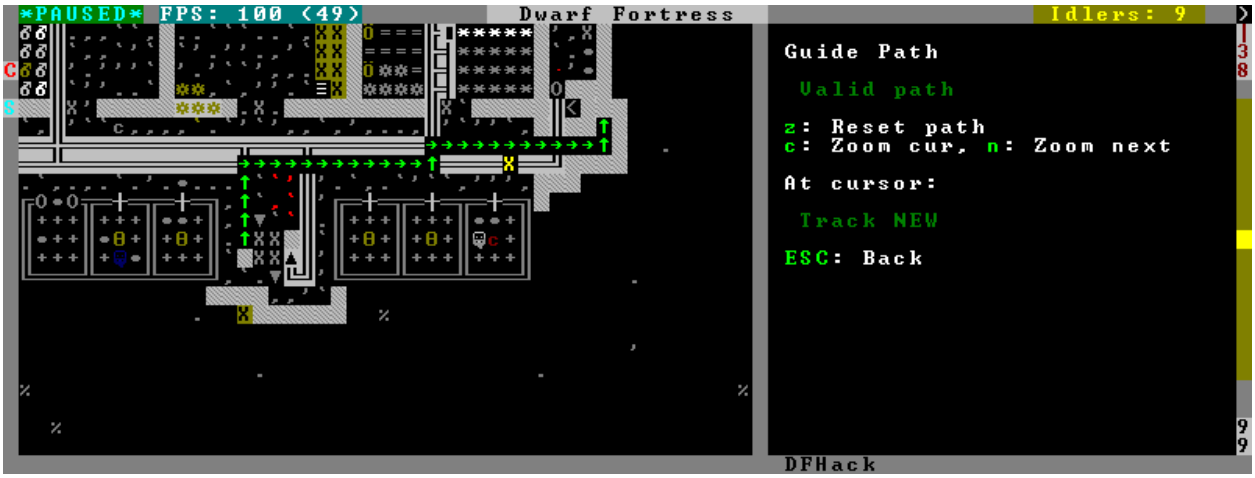
- Calling `gui/gm-editor` from a command or keybinding opens the editor on whatever is selected or viewed (e.g. unit/item description screen)
- using `gui/gm-editor <lua command>` - executes lua command and opens editor on its results (e.g. `gui/gm-editor "df.global.world.items.all"` shows all items)
- using `gui/gm-editor dialog` - shows an in game dialog to input lua command. Works the same as version above.
- using `gui/gm-editor toggle` - will hide (if shown) and show (if hidden) editor at the same position you left it

6.4.19 gui/gm-unit

An editor for various unit attributes.

6.4.20 gui/guide-path

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.

Keybinding: AltP in `dwarfmode/Hauling/DefineStop/Cond/Guide`

6.4.21 gui/hack-wish

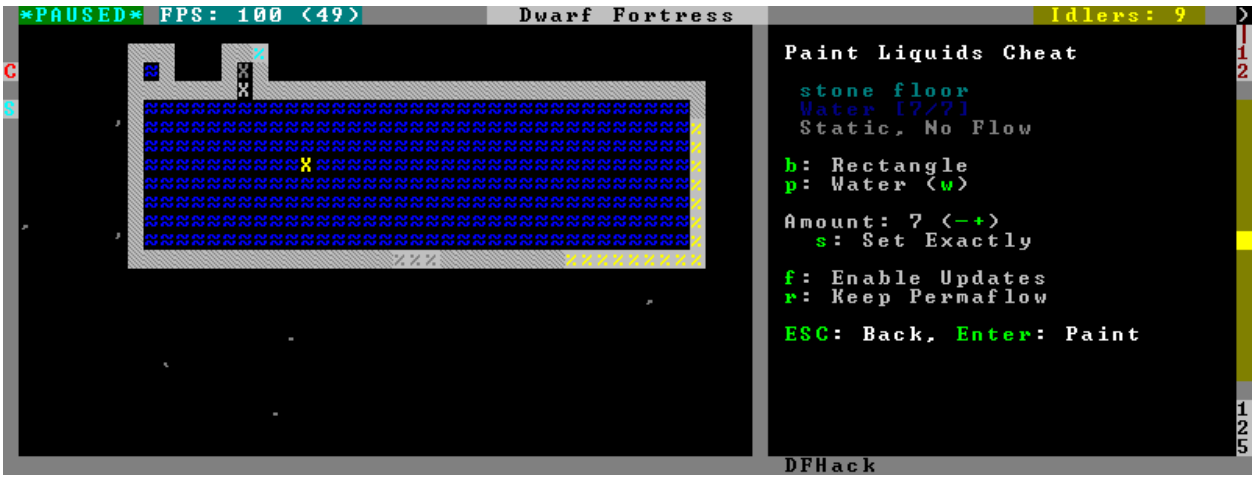
An alias for *gui/create-item*. Deprecated.

6.4.22 gui/hello-world

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

6.4.23 gui/liquids

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.

Keybinding: AltL in `dwarfmode/LookAround`

6.4.24 gui/load-screen

A replacement for the "continue game" screen.

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

The primary view is a list of saved games, much like the default list provided by DF. Several filter options are available:

- `s`: search for folder names containing specific text
- `t`: filter by active game type (e.g. fortress, adventurer)
- `b`: toggle display of backup folders, as created by DF's `AUTOBACKUP` option (see data/init/init.txt for a detailed explanation). This defaults to hiding backup folders, since they can take up significant space in the list.

When selecting a game with `Enter`, a dialog will give options to load the selected game (`Enter` again), cancel (`Esc`), or rename the game's folder (`r`). See the `title-start-rename` *tweak* to rename folders in the "start playing" menu.

6.4.25 gui/manager-quantity

Sets the quantity of the selected manager job (in the j-m or u-m screens).

Keybinding: AltQ in `jobmanagement/Main`

6.4.26 gui/mass-remove

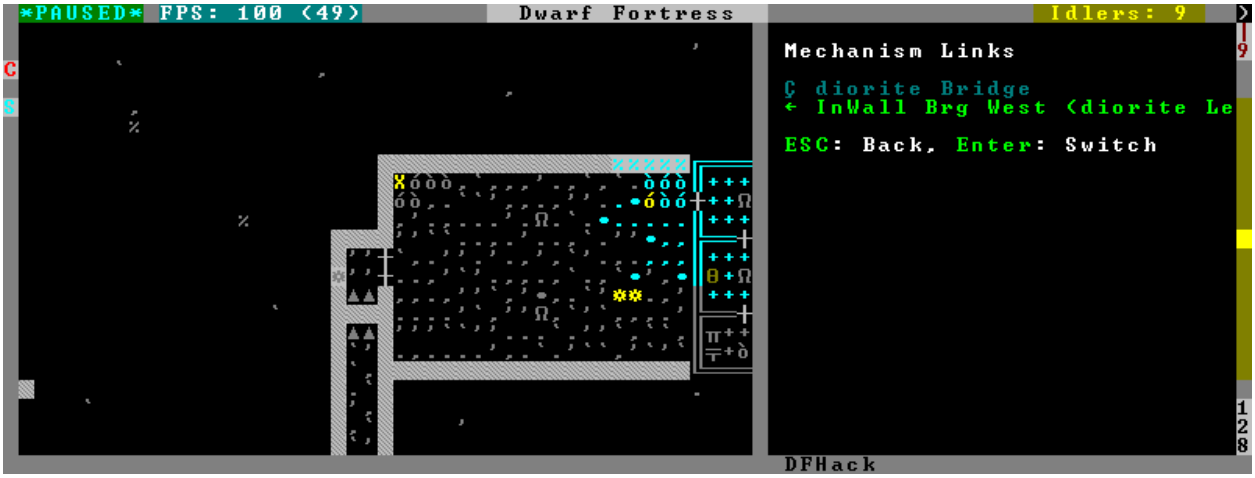
Allows removal of buildings/constructions and suspend/unsuspend using a box selection.

The following marking modes are available.

- Suspend: suspends the construction of a planned building/construction
- Unsuspend: resumes the construction of a planned building/construction
- Remove Construction: designates a construction (wall, floor, etc) for removal. Similar to the native Designate->Remove Construction menu in DF
- Unremove Construction: cancels removal of a construction (wall, floor, etc)
- Remove Building: designates a building (door, workshop, etc) for removal. Similar to the native Set Building Tasks/Prefs->Remove Building menu in DF
- Unremove Building: cancels removal of a building (door, workshop, etc)
- Remove All: designates both constructions and buildings for removal, and deletes planned buildings/constructions
- Unremove All: cancels removal designations for both constructions and buildings

6.4.27 gui/mechanisms

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

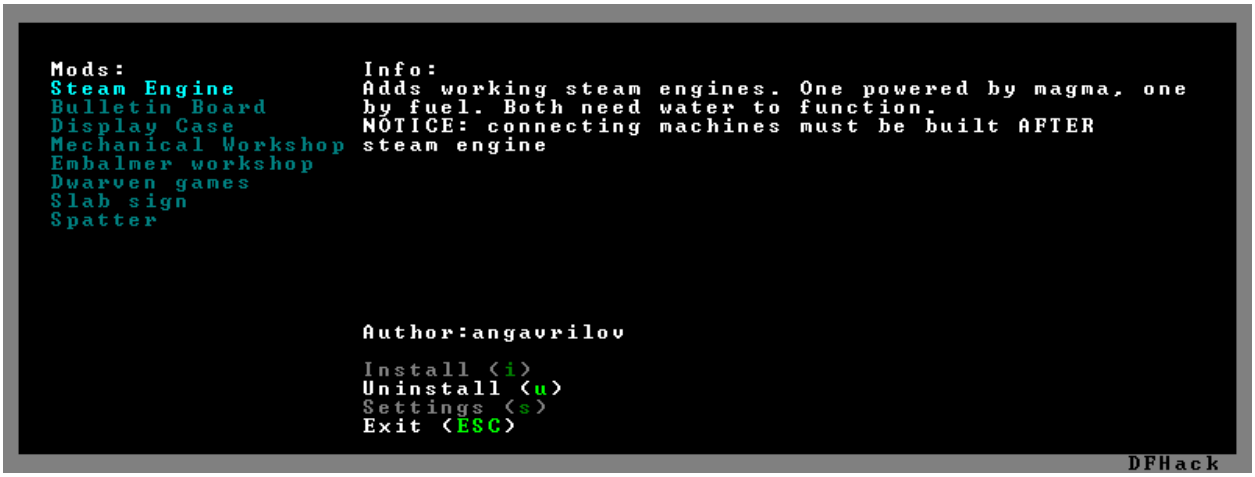


To exit, press `Esc` or `Enter`; `Esc` recenters on the original building, while `Enter` leaves focus on the current one. `ShiftEnter` has an effect equivalent to pressing `Enter`, and then re-entering the mechanisms UI.

Keybinding: CtrlM in `dwarfmode/QueryBuilding/Some`

6.4.28 gui/mod-manager

A simple way to install and remove small mods, which are not included in DFHack. Examples are available here.



Each mod is a lua script located in *<DF>*`/mods/`, which MUST define the following variables:

name a name that is displayed in list

author mod author, also displayed

description a description of the mod

Of course, this doesn't actually make a mod - so one or more of the following should also be defined:

raws_list a list (table) of file names that need to be copied over to df raws

patch_entity a chunk of text to patch entity *TODO: add settings to which entities to add*

patch_init a chunk of lua to add to lua init

patch_dofile a list (table) of files to add to lua init as "dofile"

patch_files a table of files to patch

filename a filename (in raws folder) to patch

patch what to add

after a string after which to insert

guard a token that is used in raw files to find additions and remove them on uninstall

guard_init a token for lua file

[pre|post]_(un)install Callback functions, which can trigger more complicated behavior

6.4.29 gui/no-dfhack-init

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

6.4.30 gui/pathable

Highlights each visible map tile to indicate whether it is possible to path to from the tile at the cursor - green if possible, red if not, similar to *gui/siege-engine*. A few options are available:

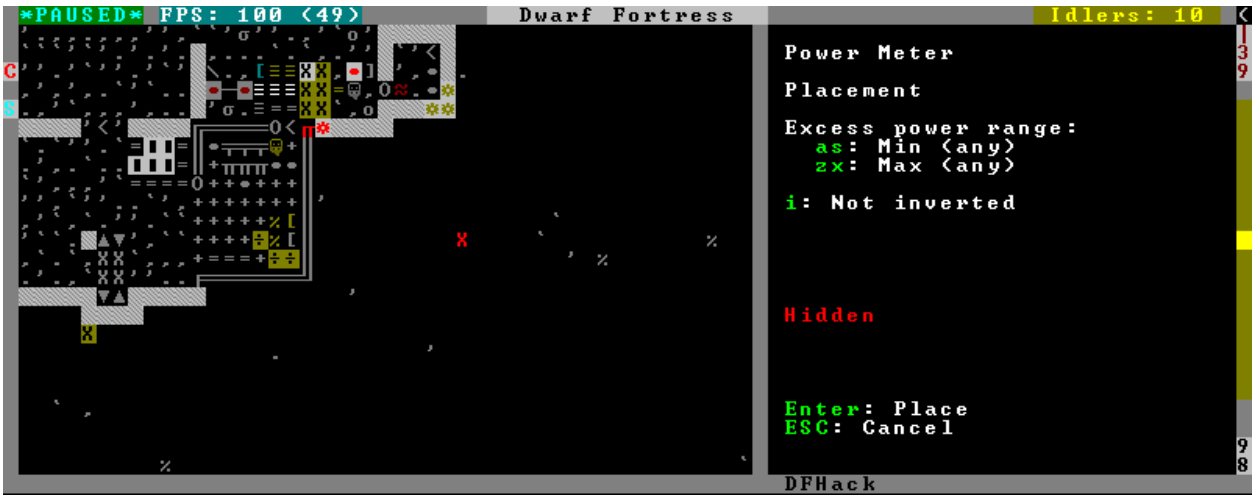
- `l`: Lock cursor: when enabled, the movement keys move around the map instead of moving the cursor. This is useful to check whether parts of the map far away from the cursor can be pathed to from the cursor.
- `d`: Draw: allows temporarily disabling the highlighting entirely.
- `u`: Skip unrevealed: when enabled, unrevealed tiles will not be highlighed at all. (These would otherwise be highlighted in red.)

Note: This tool uses a cache used by DF, which currently does *not* account for climbing. If an area of the map is only accessible by climbing, this tool may report it as inaccessible. Care should be taken when digging into the upper levels of caverns, for example.

Keybinding: AltShiftP in `dwarfmode/LookAround`

6.4.31 gui/power-meter

Activate an in-game interface for *power-meter* 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.

Keybinding: CtrlShiftM in `dwarfmode/Build/Position/Trap`

6.4.32 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.

6.4.33 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.

6.4.34 gui/quickfort

Graphical interface for the *quickfort* script. Once you load a blueprint, you will see a blinking "shadow" over the tiles that will be modified. You can use the cursor to reposition the blueprint. Once you are satisfied, hit `ENTER` to apply the blueprint to the map.

Usage:

```
gui/quickfort [<search terms>]
```

If the (optional) search terms match a single blueprint (e.g. if the search terms are copied from the `quickfort list` output like `gui/quickfort library/dreamfort.csv -n /industry1`), then that blueprint is pre-loaded into the UI and a preview for that blueprint appears. Otherwise, a dialog is shown where you can select a blueprint to load.

You can also type search terms in the dialog and the list of matching blueprints will be filtered as you type. You can search for directory names, file names, blueprint labels, modes, or comments. Note that, depending on the active list filters, the id numbers in the list may not be contiguous.

Any settings you set in the UI, such as search terms for the blueprint list, are saved between invocations.

Examples:

Command	Effect
gui/quickfort	opens the quickfort interface with saved settings
gui/quickfort dreamfort	opens with a custom blueprint filter
gui/quickfort myblueprint.csv	opens with the specified blueprint pre-loaded

Keybinding: CtrlShiftQ in `dwarfmode`

6.4.35 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.

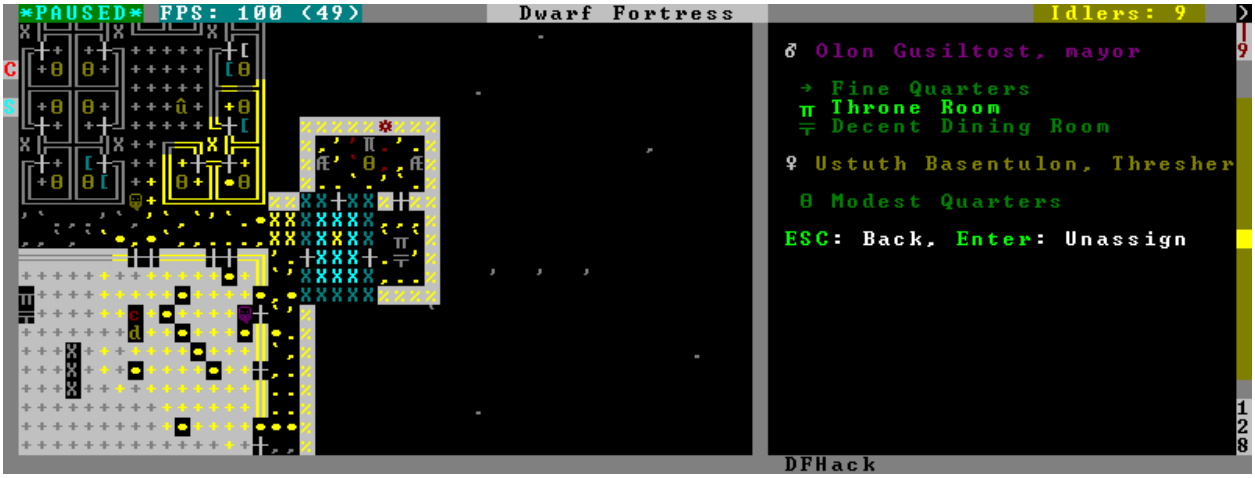
The `building` or `unit` options are automatically assumed when in relevant UI state.

Keybinding: CtrlShiftN

Keybinding: CtrlShiftT -> `"gui/rename unit-profession"`

6.4.36 gui/room-list

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.

Keybinding: AltR in `dwarfmode/QueryBuilding/Some`

6.4.37 gui/settings-manager

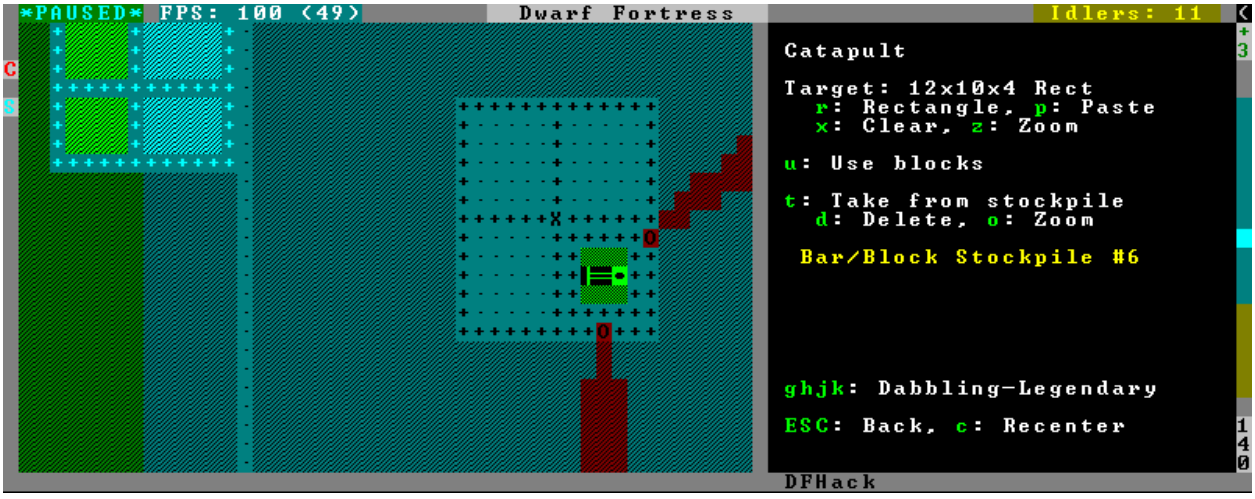
An in-game manager for settings defined in `init.txt` and `d_init.txt`.

Keybinding: AltS in `title`

Keybinding: AltS in `dwarfmode/Default`

6.4.38 gui/siege-engine

Activate an in-game interface for *siege-engine*, 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.

Keybinding: AltA in `dwarfmode/QueryBuilding/Some/SiegeEngine`

6.4.39 gui/stamper

allows manipulation of designations by transforms such as translations, reflections, rotations, and inversion. designations can also be used as brushes to erase other designations and cancel constructions.

6.4.40 gui/stockpiles

An in-game interface for *stockpiles*, 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.

Keybinding: AltL -> `"gui/stockpiles -load"` in `dwarfmode/QueryBuilding/Some/Stockpile`

Keybinding: AltS -> `"gui/stockpiles -save"` in `dwarfmode/QueryBuilding/Some/Stockpile`

6.4.41 gui/teleport

A front-end for the *teleport* script that allows choosing a unit and destination using the in-game cursor.

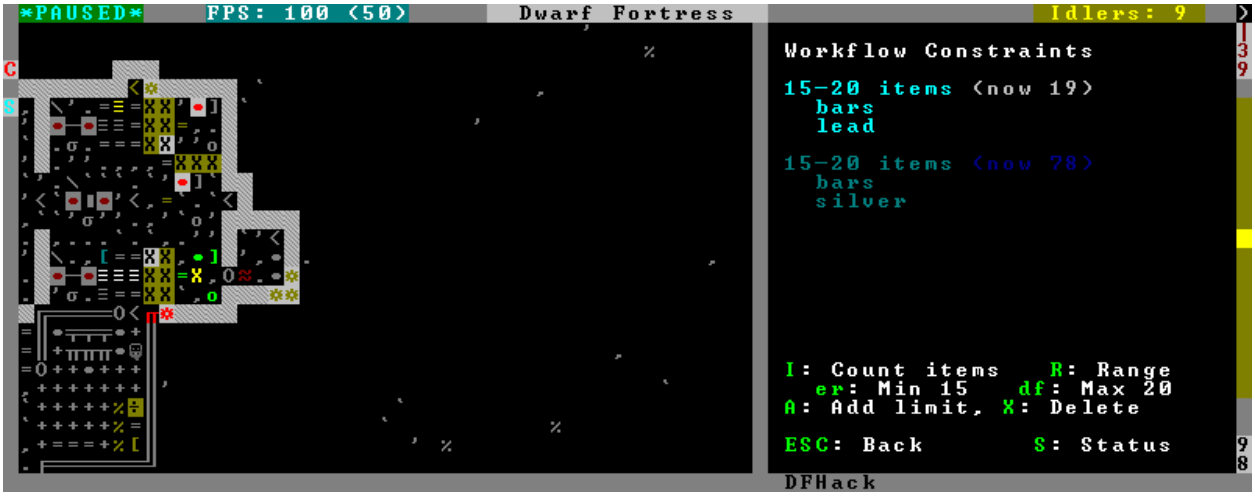
6.4.42 gui/unit-info-viewer

Displays age, birth, maxage, shearing, milking, grazing, egg laying, body size, and death info about a unit.

Keybinding: AltI in `dwarfmode/ViewUnits|unitlist`

6.4.43 gui/workflow

Bind to a key (the example config uses Alt-W), and activate with a job selected in a workshop in `q` mode.



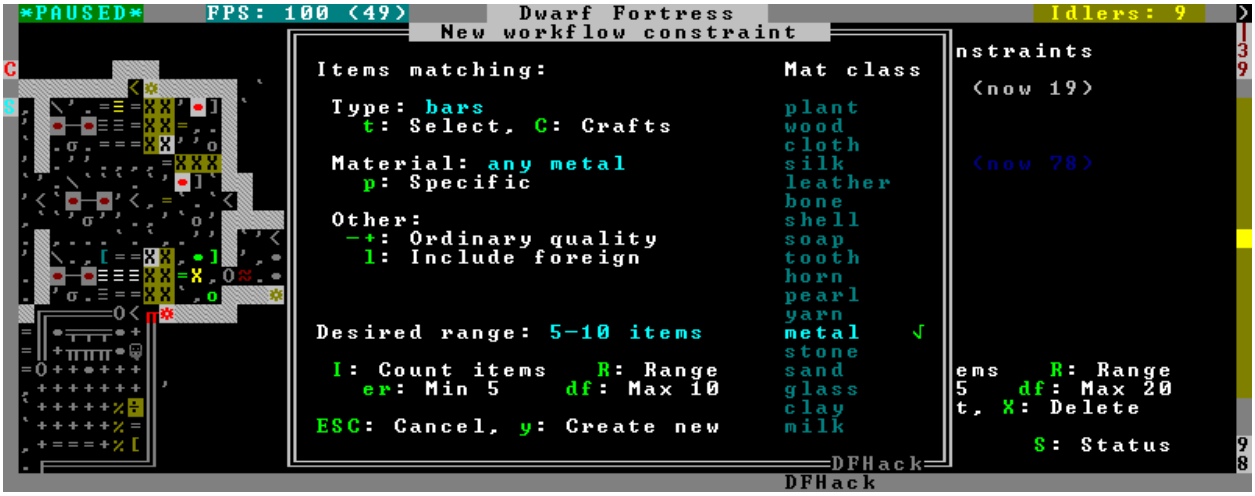
This script provides a simple interface to constraints managed by *workflow*. When active, it displays a list of all constraints applicable to the current job, and their current status.

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.

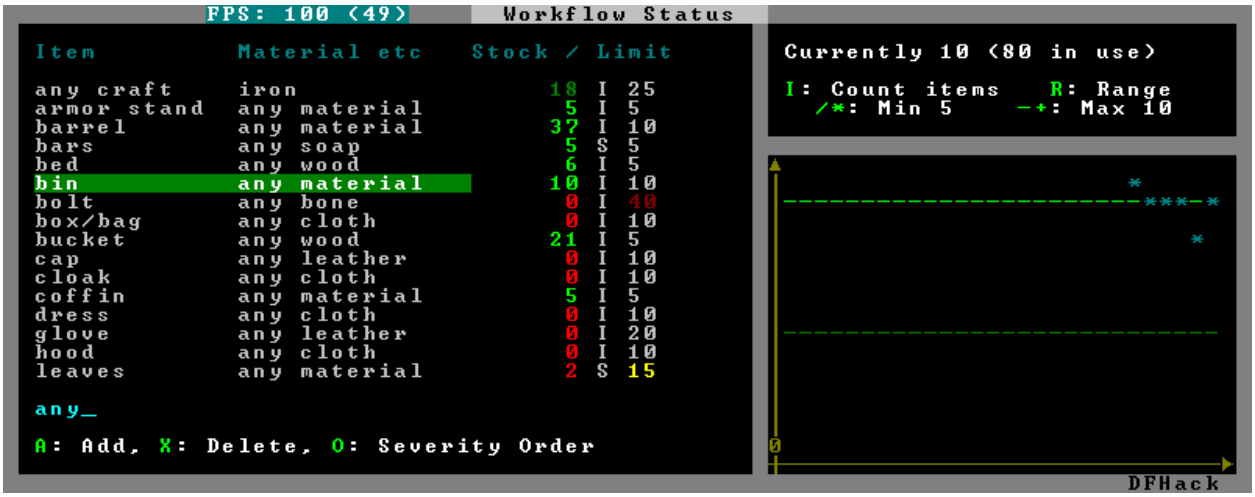
Pressing `i` switches the current constraint between counting stacks or items. Pressing `r` lets you input the range directly; `e`, `r`, `d`, `f` adjust the bounds by 5, 10, or 20 depending on the direction and the `i` setting (counting items and expanding the range each gives a 2x bonus).

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).

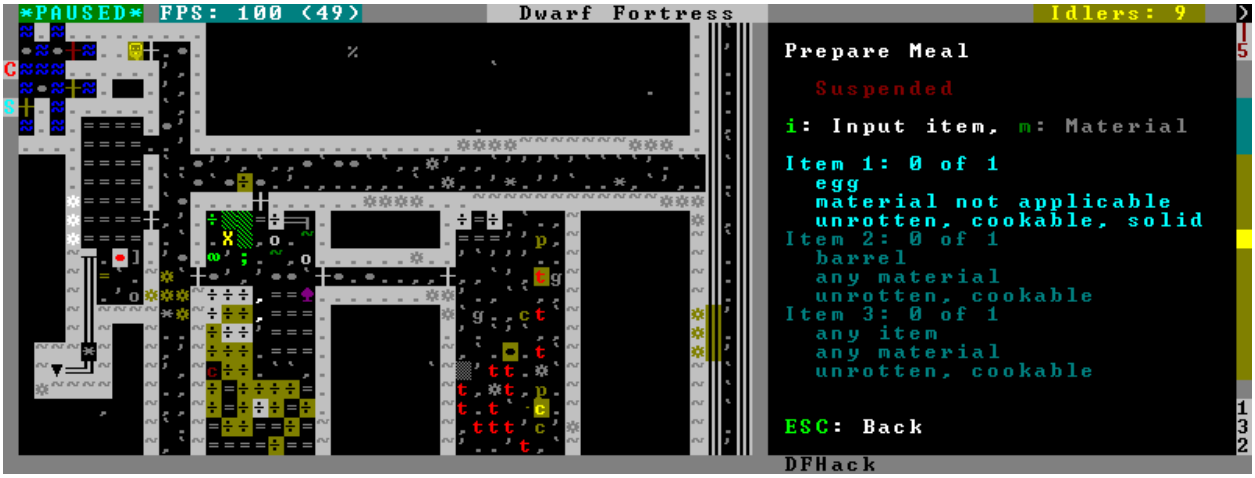
Keybinding: AltW in `dwarfmode/QueryBuilding/Some/Workshop/Job`

Keybinding: AltW -> `"gui/workflow status"` in `overallstatus`

Keybinding: AltW -> `"gui/workflow status"` in `dfhack/lua/status_overlay`

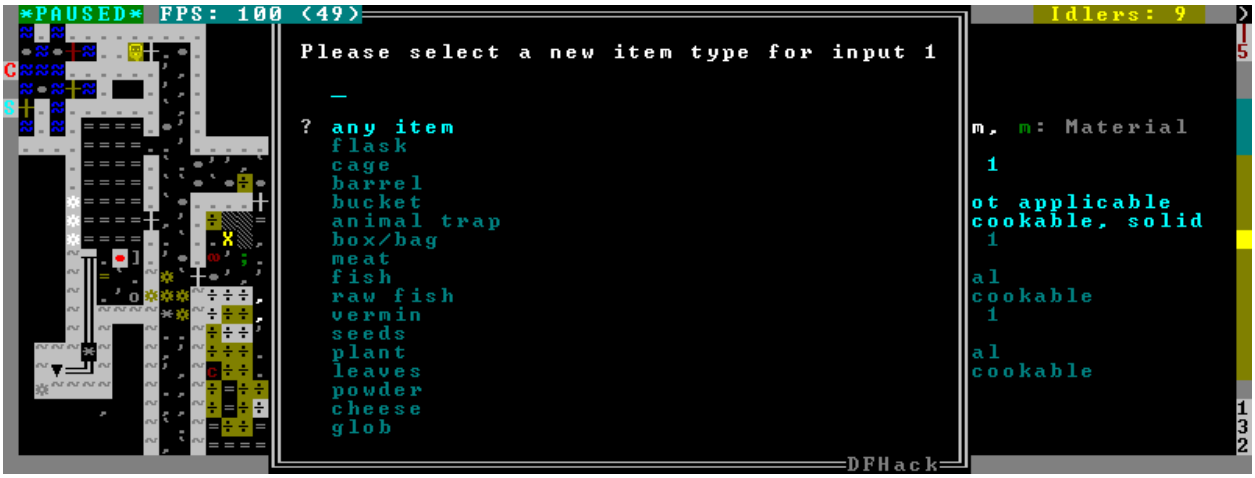
6.4.44 gui/workshop-job

Run with a job selected in a workshop in the `q` mode.

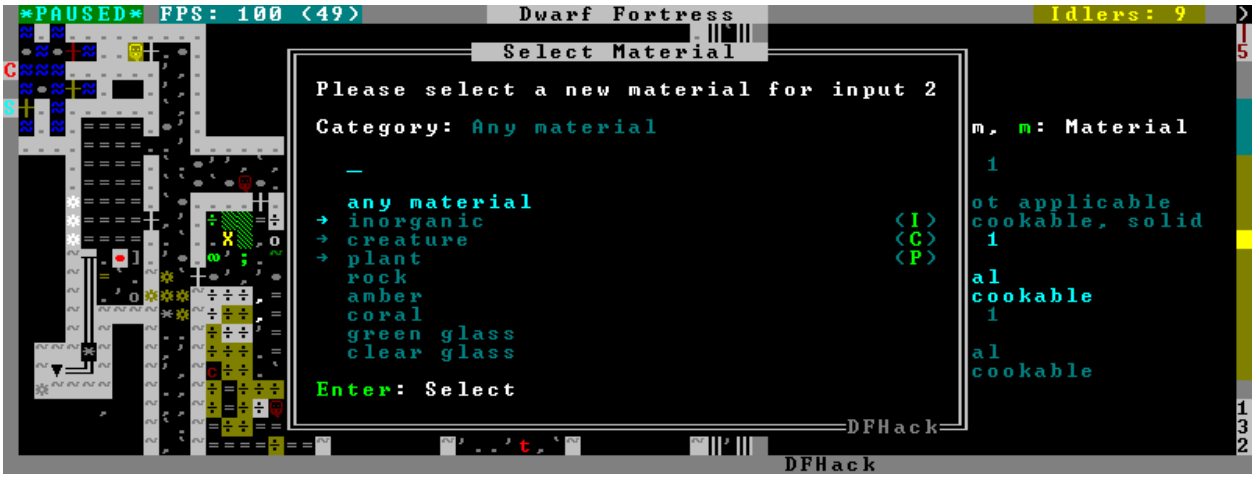


The script shows a list of the input reagents of the selected job, and allows changing them like the *job* `item-type` and *job* `item-material` commands.

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.

Keybinding: AltA in `dwarfmode/QueryBuilding/Some/Workshop/Job`

6.5 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.

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.

Argument names are preceded with a dash, and supplying the same argument name multiple times will result in an error.

The `-help` argument to any `modtools` script will print a descriptive usage string describing the arguments, similar to the documentation here.

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.

Contents

- *modtools/interaction-trigger*
- *modtools/invader-item-destroyer*
- *modtools/item-trigger*
- *modtools/moddable-gods*
- *modtools/outside-only*
- *modtools/pref-edit*
- *modtools/projectile-trigger*
- *modtools/random-trigger*
- *modtools/raw-lint*
- *modtools/reaction-product-trigger*
- *modtools/reaction-trigger*
- *modtools/reaction-trigger-transition*
- *modtools/set-belief*
- *modtools/set-need*
- *modtools/set-personality*
- *modtools/skill-change*
- *modtools/spawn-flow*
- *modtools/spawn-liquid*
- *modtools/syndrome-trigger*
- *modtools/transform-unit*

6.5.1 modtools/add-syndrome

This allows adding and removing syndromes from units.

Arguments:

```
-syndrome name|id
    the name or id of the syndrome to operate on
    examples:
        "gila monster bite"
        14
-resetPolicy policy
    specify a policy of what to do if the unit already has an
    instance of the syndrome.  examples:
        NewInstance
            default behavior: create a new instance of the syndrome
        DoNothing
        ResetDuration
        AddDuration
-erase
    instead of adding an instance of the syndrome, erase one
-eraseAll
    erase every instance of the syndrome
```

(continues on next page)

(continued from previous page)

```
-eraseClass SYN_CLASS
    erase every instance of every syndrome with the given SYN_CLASS
-target id
    the unit id of the target unit
    examples:
        0
        28
-skipImmunities
    add the syndrome to the target even if it is immune to the syndrome
```

6.5.2 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"
```

6.5.3 modtools/change-build-menu

Change the build sidebar menus.

This script provides a flexible and comprehensive system for adding and removing items from the build sidebar menus. You can add or remove workshops/furnaces by text ID, or you can add/remove ANY building via a numeric building ID triplet.

Changes made with this script do not survive a save/load. You will need to redo your changes each time the world loads.

Just to be clear: You CANNOT use this script AT ALL if there is no world loaded!

Usage:

`modtools/change-build-menu start|enable`:

`enable modtools/change-build-menu`:

> Start the ticker. This needs to be done before any changes will take effect. Note that you can make changes before or after starting the ticker, both options should work equally well.

`modtools/change-build-menu stop|disable`:

`disable modtools/change-build-menu`:

> Stop the ticker. Does not clear stored changes. The ticker will automatically stop when the current world is unloaded.

`modtools/change-build-menu add <ID> <CATEGORY> [<KEY>]`:

> Add the workshop or furnace with the ID `<ID>` to `<CATEGORY>`. `<KEY>` is an optional DF hotkey ID.
>
> **`<CATEGORY>` may be one of:**
>
> - MAIN_PAGE
> - SIEGE_ENGINES
> - TRAPS

- WORKSHOPS
- FURNACES
- CONSTRUCTIONS
- MACHINES
- CONSTRUCTIONS_TRACK

Valid `<ID>` values for hardcoded buildings are as follows:

- CARPENTERS
- FARMERS
- MASONS
- CRAFTSDWARFS
- JEWELERS
- METALSMITHSFORGE
- MAGMAFORGE
- BOWYERS
- MECHANICS
- SIEGE
- BUTCHERS
- LEATHERWORKS
- TANNERS
- CLOTHIERS
- FISHERY
- STILL
- LOOM
- QUERN
- KENNELS
- ASHERY
- KITCHEN
- DYERS
- TOOL
- MILLSTONE
- WOOD_FURNACE
- SMELTER
- GLASS_FURNACE
- MAGMA_SMELTER
- MAGMA_GLASS_FURNACE
- MAGMA_KILN

- KILN

`modtools/change-build-menu remove <ID> <CATEGORY>`:

Remove the workshop or furnace with the ID `<ID>` from `<CATEGORY>`.

`<CATEGORY>` and `<ID>` may have the same values as for the "add" option.

`modtools/change-build-menu revert <ID> <CATEGORY>`:

Revert an earlier remove or add operation. It is NOT safe to "remove" an "add"ed building or vice versa, use this option to reverse any changes you no longer want/need.

Module Usage:

To use this script as a module put the following somewhere in your own script:

```
local buildmenu = reqscript "change-build-menu"
```

Then you can call the functions documented here like so:

- Example: Remove the carpenters workshop:

```
buildmenu.ChangeBuilding("CARPENTERS", "WORKSHOPS", false)
```

- Example: Make it impossible to build walls (not recommended!):

```
local typ, styp = df.building_type.Construction, df.construction_type.Wall
buildmenu.ChangeBuildingAdv(typ, styp, -1, "CONSTRUCTIONS", false)
```

Note that to allow any of your changes to take effect you need to start the ticker. See the "Command Usage" section.

Global Functions:

`GetWShopID(btype, bsubtype, bcustom)`: GetWShopID returns a workshop's or furnace's string ID based on its numeric ID triplet. This string ID *should* match what is expected by eventful for hardcoded buildings.

`GetWShopType(id)`: GetWShopIDs returns a workshop or furnace's ID numbers as a table. The passed in ID should be the building's string identifier, it makes no difference if it is a custom building or a hardcoded one. The return table is structured like so: `{type, subtype, custom}`

`IsEntityPermitted(id)`: IsEntityPermitted returns true if DF would normally allow you to build a workshop or furnace. Use this if you want to change a building, but only if it is permitted in the current entity. You do not need to specify an entity, the current fortress race is used.

`ChangeBuilding(id, category, [add, [key]])`:

`ChangeBuildingAdv(typ, subtyp, custom, category, [add, [key]])`: These two functions apply changes to the build sidebar menus. If "add" is true then the building is added to the specified category, else it is removed. When adding you may specify "key", a string DF hotkey ID.

The first version of this function takes a workshop or furnace ID as a string, the second takes a numeric ID triplet (which can specify any building, not just workshops or furnaces).

`RevertBuildingChanges(id, category)`:

`RevertBuildingChangesAdv(typ, subtyp, custom, category)`: These two functions revert changes made by "ChangeBuilding" and "ChangeBuildingAdv". Like those two functions there are two versions, a simple one that takes a string ID and one that takes a numeric ID triplet.

6.5.4 modtools/create-item

Replaces the *createitem* plugin, with standard arguments. The other versions will be phased out in a later version.

Arguments:

```
-creator id
    specify the id of the unit who will create the item,
    or \\LAST to indicate the unit with id df.global.unit_next_id-1
    examples:
        0
        2
        \\LAST
-material matstring
    specify the material of the item to be created
    examples:
        INORGANIC:IRON
        CREATURE_MAT:DWARF:BRAIN
        PLANT_MAT:MUSHROOM_HELMET_PLUMP:DRINK
-item itemstr
    specify the itemdef of the item to be created
    examples:
        WEAPON:ITEM_WEAPON_PICK
-quality qualitystr
    specify the quality level of the item to be created (df.item_quality)
    examples: Ordinary, WellCrafted, FinelyCrafted, Masterful, or 0-5
-matchingShoes
    create two of this item
-matchingGloves
    create two of this item, and set handedness appropriately
```

6.5.5 modtools/create-tree

Spawns a tree.

Usage:

```
-tree treeName
    specify the tree to be created
    examples:
        OAK
        NETHER_CAP

-age howOld
    set the age of the tree in years (integers only)
    defaults to 1 if omitted

-location [ x y z ]
    create the tree at the specified coordinates

example:
    modtools/create-tree -tree OAK -age 100 -location [ 33 145 137 ]
```

6.5.6 modtools/create-unit

Creates a unit. Usage:

```
-race raceName
    (obligatory)
    Specify the race of the unit to be created.
    examples:
        DWARF
        HUMAN

-caste casteName
    Specify the caste of the unit to be created.
    If omitted, the caste is randomly selected.
    examples:
        MALE
        FEMALE
        DEFAULT

-domesticate
    Tames the unit if it lacks the CAN_LEARN and CAN_SPEAK tokens.

-civId id
    Make the created unit a member of the specified civilisation
    (or none if id = -1).  If id is \\LOCAL, make it a member of the
    civ associated with the fort; otherwise id must be an integer

-groupId id
    Make the created unit a member of the specified group
    (or none if id = -1).  If id is \\LOCAL, make it a member of the
    group associated with the fort; otherwise id must be an integer

-setUnitToFort
    Sets the groupId and civId to those of the player in Fortress mode.
    Equivalent to -civId \\LOCAL and -groupId \\LOCAL.

-name entityRawName
    Set the unit's name to be a random name appropriate for the
    given entity. \\LOCAL can be specified instead to automatically
    use the fort group entity in fortress mode only.
    examples:
        MOUNTAIN
        EVIL

-nick nickname
    This can be included to nickname the unit.
    Replace "nickname" with the desired name.

-age howOld
    This can be included to specify the unit's age.
    Replace "howOld" with a (non-negative) number.
    The unit's age is set randomly if this is omitted.

-equip [ ITEM:MATERIAL:QUANTITY ... ]
    This can be included to create items and equip them onto
        the created unit.
    This is carried out via the same logic used in arena mode,
        so equipment will always be sized correctly and placed
        on what the game deems to be appropriate bodyparts.
        Clothing is also layered in the appropriate order.
    Note that this currently comes with some limitations,
```

(continues on next page)

(continued from previous page)

```
        such as an inability to specify item quality
        and objects not being placed in containers
        (for example, arrows are not placed in quivers).
    Item quantity defaults to 1 if omitted.
    When spaces are included in the item or material name,
        the entire item description should be enclosed in
        quotation marks. This can also be done to increase
        legibility when specifying multiple items.
    examples:
        -equip [ RING:CREATURE:DWARF:BONE:3 ]
            3 dwarf bone rings
        -equip [ ITEM_WEAPON_PICK:INORGANIC:IRON ]
            1 iron pick
        -equip [ "ITEM_SHIELD_BUCKLER:PLANT:OAK:WOOD" "AMULET:AMBER" ]
            1 oaken buckler and 1 amber amulet

-skills [ SKILL:LEVEL ... ]
    This can be included to add skills to the created unit.
    Specify a skill token followed by a skill level value.
    Look up "Skill Token" and "Skill" on the DF Wiki for a list
        of valid tokens and levels respectively.
    Note that the skill level provided must be a number greater than 0.
    If the unit possesses a matching natural skill, this is added to it.
    Quotation marks can be added for legibility as explained above.
    example:
        -skill [ SNEAK:1 EXTRACT_STRAND:15 ]
            novice ambusher, legendary strand extractor

-profession token
    This can be included to set the unit's profession.
    Replace "token" with a Unit Type Token (check the DF Wiki for a list).
    For skill-based professions, it is recommended to give the unit
        the appropriate skill set via -skills.
    This can also be used to make animals trained for war/hunting.
    Note that this will be overridden if the unit has been given the age
        of a baby or child, as these have a special "profession" set.
    Using this for setting baby/child status is not recommended;
        this should be done via -age instead.
    examples:
        STRAND_EXTRACTOR
        MASTER_SWORDSMAN
        TRAINED_WAR

-customProfession name
    This can be included to give the unit a custom profession name.
    Enclose the name in quotation marks if it includes spaces.
    example:
        -customProfession "Destroyer of Worlds"

-duration ticks
    If this is included, the unit will vanish in a puff of smoke
        once the specified number of ticks has elapsed.
    Replace "ticks" with an integer greater than 0.
    Note that the unit's equipment will not vanish.

-quantity howMany
    This can be included to create multiple creatures simultaneously.
```

(continues on next page)

(continued from previous page)

```
    Replace "howMany" with the desired number of creatures.
    Quantity defaults to 1 if this is omitted.

-location [ x y z ]
    (obligatory)
    Specify the coordinates where you want the unit to appear.

-locationRange [ x_offset y_offset z_offset ]
    If included, the unit will be spawned at a random location
        within the specified range relative to the target -location.
    z_offset defaults to 0 if omitted.
    When creating multiple units, the location is randomised each time.
    example:
        -locationRange [ 4 3 1 ]
            attempts to place the unit anywhere within
            -4 to +4 tiles on the x-axis
            -3 to +3 tiles on the y-axis
            -1 to +1 tiles on the z-axis
            from the specified -location coordinates

-locationType type
    May be used with -locationRange
        to specify what counts as a valid tile for unit spawning.
    Unit creation will not occur if no valid tiles are available.
    Replace "type" with one of the following:
        Walkable
            units will only be placed on walkable ground tiles
            this is the default used if -locationType is omitted
        Open
            open spaces are also valid spawn points
            this is intended for flying units
        Any
            all tiles, including solid walls, are valid
            this is only recommended for ghosts not carrying items

-flagSet [ flag1 flag2 ... ]
    This can be used to set the specified unit flags to true.
    Flags may be selected from:
        df.unit_flags1
        df.unit_flags2
        df.unit_flags3
        df.unit_flags4
    example:
        flagSet [ announce_titan ]
            causes an announcement describing the unit to appear
            when it is discovered ("[Unit] has come! ...")

-flagClear [ flag1 flag2 ... ]
    As above, but sets the specified unit flags to false.
```

6.5.7 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*.

6.5.8 modtools/extra-gamelog

This script writes extra information to the gamelog. This is useful for tools like Soundsense.

Usage:

```
modtools/extra-gamelog enable
modtools/extra-gamelog disable
```

6.5.9 modtools/force

This tool triggers events like megabeasts, caravans, and migrants.

Usage:

```
-eventType event
    specify the type of the event to trigger
    examples:
        Megabeast
        Migrants
        Caravan
        Diplomat
        WildlifeCurious
        WildlifeMischievous
        WildlifeFlier
        NightCreature
-civ entity
    specify the civ of the event, if applicable
    examples:
        player
        MOUNTAIN
        EVIL
        28
```

6.5.10 modtools/if-entity

Run a command if the current entity matches a given ID.

To use this script effectively it needs to be called from "raw/onload.init". Calling this from the main dfhack.init file will do nothing, as no world has been loaded yet.

Usage:

- **`id:`** Specify the entity ID to match
- **`cmd [ commandStrs ]:`** Specify the command to be run if the current entity matches the entity given via -id

All arguments are required.

Example:

- Print a message if you load an elf fort, but not a dwarf, human, etc. fort:

```
if-entity -id "FOREST" -cmd [ lua "print('Dirty hippies.')" ]
```

6.5.11 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.

Usage:

```
-clear
    unregisters all triggers
-onAttackStr str
    trigger the command when the attack verb is "str". both onAttackStr and␣
↪onDefendStr MUST be specified
-onDefendStr str
    trigger the command when the defend verb is "str". both onAttackStr and␣
↪onDefendStr MUST be specified
-suppressAttack
    delete the attack announcement from the combat logs
-suppressDefend
    delete the defend announcement from the combat logs
-command [ commandStrs ]
    specify the command to be executed
    commandStrs
        \\ATTACK_VERB
        \\DEFEND_VERB
        \\ATTACKER_ID
        \\DEFENDER_ID
        \\ATTACK_REPORT
        \\DEFEND_REPORT
        \\anything -> \anything
        anything -> anything
```

You must specify both an attack string and a defend string to guarantee correct performance. Either will trigger the script when it happens, but it will not be triggered twice in a row if both happen.

6.5.12 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.

Arguments:

```
-clear
    reset all registered data
-allEntities [true/false]
    set whether it should delete items from invaders from any civ
-allItems [true/false]
    set whether it should delete all invader items regardless of
    type when an appropriate invader dies
-item itemdef
    set a particular itemdef to be destroyed when an invader
    from an appropriate civ dies.  examples:
        ITEM_WEAPON_PICK
-entity entityName
    set a particular entity up so that its invaders destroy their
    items shortly after death.  examples:
```

(continues on next page)

(continued from previous page)

```
        MOUNTAIN
        EVIL
```

6.5.13 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.

Arguments:

```
-clear
    clear all registered triggers
-checkAttackEvery n
    check the attack event at least every n ticks
-checkInventoryEvery n
    check inventory event at least every n ticks
-itemType type
    trigger the command for items of this type
    examples:
        ITEM_WEAPON_PICK
        RING
-onStrike
    trigger the command on appropriate weapon strikes
-onEquip mode
    trigger the command when someone equips an appropriate item
    Optionally, the equipment mode can be specified
    Possible values for mode:
        Hauled
        Weapon
        Worn
        Piercing
        Flask
        WrappedAround
        StuckIn
        InMouth
        Pet
        SewnInto
        Strapped
    multiple values can be specified simultaneously
    example: -onEquip [ Weapon Worn Hauled ]
-onUnequip mode
    trigger the command when someone unequips an appropriate item
    see above note regarding 'mode' values
-material mat
    trigger the commmand on items with the given material
    examples
        INORGANIC:IRON
        CREATURE:DWARF:BRAIN
        PLANT:OAK:WOOD
-contaminant mat
    trigger the command for items with a given material contaminant
    examples
        INORGANIC:GOLD
        CREATURE:HUMAN:BLOOD
        PLANT:MUSHROOM_HELMET_PLUMP:DRINK
```

(continues on next page)

(continued from previous page)

```
        WATER
-command [ commandStrs ]
    specify the command to be executed
    commandStrs
        \\ATTACKER_ID
        \\DEFENDER_ID
        \\ITEM_MATERIAL
        \\ITEM_MATERIAL_TYPE
        \\ITEM_ID
        \\ITEM_TYPE
        \\CONTAMINANT_MATERIAL
        \\CONTAMINANT_MATERIAL_TYPE
        \\CONTAMINANT_MATERIAL_INDEX
        \\MODE
        \\UNIT_ID
        \\anything -> \anything
        anything -> anything
```

6.5.14 modtools/moddable-gods

This is a standardized version of Putnam's moddableGods script. It allows you to create gods on the command-line.

Arguments:

```
-name godName
    sets the name of the god to godName
    if there's already a god of that name, the script halts
-spheres [ sphereList ]
    define a space-separated list of spheres of influence of the god
-gender male|female|neuter
    sets the gender of the god
-depictedAs str
    often depicted as a str
-verbose
    if specified, prints details about the created god
```

6.5.15 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.

Arguments:

```
-clear
    clears the list of registered buildings
-checkEvery n
    set how often existing buildings are checked for whether they
    are in the appropriate location to n ticks
-type [EITHER, OUTSIDE_ONLY, INSIDE_ONLY]
    specify what sort of restriction to put on the building
-building name
    specify the id of the building
```

6.5.16 modtools/pref-edit

Add, remove, or edit the preferences of a unit. Requires a modifier, a unit argument, and filters.

- **-unit <UNIT ID>:** The given unit will be affected. If not found/provided, the script will try defaulting to the currently selected unit.

Valid modifiers:

- **-add:** Add a new preference to the unit. Filters describe the preference's variables.
- **-remove:** Remove a preference from the unit. Filters describe what preference to remove.
- **-has:** Checks if the unit has a preference matching the filters. Prints a message in the console.
- **-removeall:** Remove all preferences from the unit. Doesn't require any filters.

Valid filters:

- **-id <VALUE>:** This is the ID used for all preferences that require an ID. Represents item_type, creature_id, color_id, shape_id, plant_id, poetic_form_id, musical_form_id, and dance_form_id. Text IDs (e.g. "TOAD", "AMBER") can be used for all but poetic, musical, and dance.
- **-item, -creature, -color, -shape, -plant, -poetic, -musical, -dance:** Include one of these to describe what the id argument represents.
- **-type <PREFERENCE TYPE>:** This describes the type of the preference. Can be entered either using the numerical ID or text id. Run `lua @df.unit_preference.T_type` for a full list of valid values.
- **-subtype <ID>:** The value for an item's subtype
- **-material <ID>:** The id of the material. For example "MUSHROOM_HELMET_PLUMP:DRINK" or "INORGANIC:IRON".
- **-state <STATE ID>:** The state of the material. Values can be the numerical or text ID. Run `lua @df.matter_state` for a full list of valid values.
- **-active <TRUE/FALSE>:** Whether the preference is active or not (?)

Other arguments:

- **-help:** Shows this help page.

Example usage:

- Like drinking dwarf blood:

```
modtools/pref-edit -add -item -id DRINK -material DWARF:BLOOD -type LikeFood
```

6.5.17 modtools/projectile-trigger

This triggers dfhack commands when projectiles hit their targets. Usage:

```
-clear
    unregister all triggers
-material
    specify a material for projectiles that will trigger the command
    examples:
        INORGANIC:IRON
        CREATURE_MAT:DWARF:BRAIN
        PLANT_MAT:MUSHROOM_HELMET_PLUMP:DRINK
-command [ commandList ]
```

(continues on next page)

(continued from previous page)

```
    \\LOCATION
    \\PROJECTILE_ID
    \\FIRER_ID
    \\anything -> \anything
    anything -> anything
```

6.5.18 modtools/random-trigger

Trigger 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.

Events are mutually-exclusive - register a list of scripts along with relative weights, then tell the script to select and run one with the specified probabilities. The weights must be positive integers, but they do NOT have to sum to any particular number.

The outcomes are mutually exclusive: only one will be triggered. If you want multiple independent random events, call the script multiple times.

99% of the time, you won't need to worry about this, but just in case, you can specify a name of a list of outcomes to prevent interference from other scripts that call this one. That also permits situations where you don't know until runtime what outcomes you want. For example, you could make a *modtools/reaction-trigger* that registers the worker as a mayor candidate, then run this script to choose a random mayor from the list of units that did the mayor reaction.

Arguments:

```
-outcomeListName name
    specify the name of this list of outcomes to prevent interference
    if two scripts are registering outcomes at the same time.  If none
    is specified, the default outcome list is selected automatically.
-command [ commandStrs ]
    specify the command to be run if this outcome is selected
    must be specified unless the -trigger argument is given
-weight n
    the relative probability weight of this outcome
    n must be a non-negative integer
    if not specified, n=1 is used by default
-trigger
    selects a random script based on the specified outcomeList
    (or the default one if none is specified)
-preserveList
    when combined with trigger, preserves the list of outcomes so you
    don't have to register them again.
-withProbability p
    p is a real number between 0 and 1 inclusive
    triggers the command immediately with this probability
-seed s
    sets the random seed for debugging purposes
    (guarantees the same sequence of random numbers will be produced)
    use
-listOutcomes
    lists the currently registered list of outcomes of the outcomeList
    along with their probability weights, for debugging purposes
-clear
    unregister everything
```

Note: `-preserveList` is something of a beta feature, which should be avoided by users without a specific reason to use it.

It is highly recommended that you always specify `-outcomeListName` when you give this command to prevent almost certain interference. If you want to trigger one of 5 outcomes three times, you might want this option even without `-outcomeListName`.

The list is NOT retained across game save/load, as nobody has yet had a use for this feature. Contact expwnent if you would use it; it's not that hard but if nobody wants it he won't bother.

6.5.19 modtools/raw-lint

Checks for simple issues with raw files. Can be run automatically.

6.5.20 modtools/reaction-product-trigger

This triggers dfhack commands when reaction products are produced, once per product. Usage:

```
-clear
    unregister all reaction hooks
-reactionName name
    specify the name of the reaction
-command [ commandStrs ]
    specify the command to be run on the target(s)
    special args
        \\WORKER_ID
        \\REACTION
        \\BUILDING_ID
        \\LOCATION
        \\INPUT_ITEMS
        \\OUTPUT_ITEMS
        \\anything -> \anything
        anything -> anything
```

6.5.21 modtools/reaction-trigger

Triggers dfhack commands when custom reactions complete, regardless of whether it produced anything, once per completion. Arguments:

```
-clear
    unregister all reaction hooks
-reactionName name
    specify the name of the reaction
-syndrome name
    specify the name of the syndrome to be applied to valid targets
-allowNonworkerTargets
    allow other units to be targeted if the worker is invalid or ignored
-allowMultipleTargets
    allow all valid targets within range to be affected
    if absent:
        if running a script, only one target will be used
        if applying a syndrome, then only one target will be infected
```

(continues on next page)

(continued from previous page)

```
-ignoreWorker
    ignores the worker when selecting the targets
-dontSkipInactive
    when selecting targets in range, include creatures that are inactive
    dead creatures count as inactive
-range [ x y z ]
    controls how far elligible targets can be from the workshop
    defaults to [ 0 0 0 ] (on a workshop tile)
    negative numbers can be used to ignore outer squares of the workshop
    line of sight is not respected, and the worker is always within range
-resetPolicy policy
    the policy in the case that the syndrome is already present
    policy
        NewInstance (default)
        DoNothing
        ResetDuration
        AddDuration
-command [ commandStrs ]
    specify the command to be run on the target(s)
    special args
        \\WORKER_ID
        \\TARGET_ID
        \\BUILDING_ID
        \\LOCATION
        \\REACTION_NAME
        \\anything -> \anything
        anything -> anything
    when used with -syndrome, the target must be valid for the syndrome
    otherwise, the command will not be run for that target
```

6.5.22 modtools/reaction-trigger-transition

Prints useful things to the console and a file to help modders transition from `autoSyndrome` to *modtools/reaction-trigger*.

This script is basically an apology for breaking backward compatibility in June 2014, and will be removed eventually.

6.5.23 modtools/set-belief

Changes the beliefs (values) of units. Requires a belief, modifier, and a target.

Valid beliefs:

all Apply the edit to all the target's beliefs

belief <ID> ID of the belief to edit. For example, 0 or LAW.

Valid modifiers:

set <-50-50> Set belief to given strength.

tier <1-7> Set belief to within the bounds of a strength tier:

Value	Strength
1	Lowest
2	Very Low
3	Low
4	Neutral
5	High
6	Very High
7	Highest

modify <amount> Modify current belief strength by given amount. Negative values need a \ before the negative symbol e.g. `\-1`

step <amount> Modify current belief tier up/down by given amount. Negative values need a \ before the negative symbol e.g. `\-1`

random Use the default probabilities to set the belief to a new random value.

default Belief will be set to cultural default.

Valid targets:

citizens All (sane) citizens of your fort will be affected. Will do nothing in adventure mode.

unit <UNIT ID> The given unit will be affected.

If no target is given, the provided unit can't be found, or no unit id is given with the unit argument, the script will try and default to targeting the currently selected unit.

Other arguments:

help Shows this help page.

list Prints a list of all beliefs + their IDs.

noneed By default, unit's needs will be recalculated to reflect new beliefs after every run. Use this argument to disable that functionality.

listunit Prints a list of all a unit's beliefs. Cultural defaults are marked with `*`.

6.5.24 modtools/set-need

Sets and edits unit needs.

Valid commands:

add Add a new need to the unit. Requires a -need argument, and target. -focus and -level can be used to set starting values, otherwise they'll fall back to defaults.

remove Remove an existing need from the unit. Requires a need target, and target.

edit Change an existing need in some way. Requires a need target, at least one effect, and a target.

revert Revert a unit's needs list back to its original selection and need strengths. Focus levels are preserved if the unit has a need before and after. Requires a target.

Valid need targets:

need <ID> ID of the need to target. For example 0 or DrinkAlcohol. If the need is PrayOrMedidate, a -deity argument is also required.

deity <HISTFIG ID> Required when using PrayOrMedidate needs. This value should be the historical figure ID of the deity in question.

all All of the target's needs will be affected.

Valid effects:

focus <NUMBER> Set the focus level of the targeted need. 400 is the value used when a need has just been satisfied.

level <NUMBER> Set the need level of the targeted need. Default game values are: 1 (Slight need), 2 (Moderate need), 5 (Strong need), 10 (Intense need)

Valid targets:

citizens All (sane) citizens of your fort will be affected. Will do nothing in adventure mode.

unit <UNIT ID> The given unit will be affected.

If no target is given, the provided unit can't be found, or no unit id is given with the unit argument, the script will try and default to targeting the currently selected unit.

Other arguments:

help Shows this help page.

list Prints a list of all needs + their IDs.

listunit Prints a list of all a unit's needs, their strengths, and their current focus.

Usage example - Satisfy all citizen's needs:

```
modtools/set-need -edit -all -focus 400 -citizens
```

6.5.25 modtools/set-personality

Changes the personality of units. Requires a trait, modifier, and a target.

Valid traits:

all Apply the edit to all the target's traits

trait <ID> ID of the trait to edit. For example, 0 or HATE_PROPENSITY.

Valid modifiers:

set <0-100> Set trait to given strength.

tier <1-7> Set trait to within the bounds of a strength tier.

Value	Strength
1	Lowest
2	Very Low
3	Low
4	Neutral
5	High
6	Very High
7	Highest

modify <amount> Modify current base trait strength by given amount. Negative values need a \ before the negative symbol e.g. `\-1`

step <amount> Modify current trait tier up/down by given amount. Negative values need a \ before the negative symbol e.g. `\-1`

random Set the trait to a new random value.

average Sets trait to the creature's caste's average value (as defined in the PERSONALITY creature tokens).

Valid targets:

citizens All (sane) citizens of your fort will be affected. Will do nothing in adventure mode.

unit <UNIT ID> The given unit will be affected.

If no target is given, the provided unit can't be found, or no unit id is given with the unit argument, the script will try and default to targeting the currently selected unit.

Other arguments:

help Shows this help page.

list Prints a list of all facets + their IDs.

noneed By default, unit's needs will be recalculated to reflect new traits after every run. Use this argument to disable that functionality.

listunit Prints a list of all a unit's personality traits, with their modified trait value in brackets.

6.5.26 modtools/skill-change

Sets or modifies a skill of a unit. Args:

-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

-loud if present, prints changes to console

6.5.27 modtools/spawn-flow

Creates flows at the specified location.

Arguments:

```
-material mat
    specify the material of the flow, if applicable
    examples:
        INORGANIC:IRON
        CREATURE_MAT:DWARF:BRAIN
        PLANT_MAT:MUSHROOM_HELMET_PLUMP:DRINK
-location [ x y z]
    the location to spawn the flow
-flowType type
    specify the flow type
    examples:
        Miasma
        Steam
        Mist
```

(continues on next page)

(continued from previous page)

```
        MaterialDust
        MagmaMist
        Smoke
        Dragonfire
        Fire
        Web
        MaterialGas
        MaterialVapor
        OceanWave
        SeaFoam
-flowSize size
    specify how big the flow is
```

6.5.28 modtools/spawn-liquid

This script spawns liquid at the given coordinates.

Run `modtools/spawn-liquid help` for usage.

6.5.29 modtools/syndrome-trigger

Triggers dfhack commands when syndromes are applied to units.

Arguments:

```
-clear
    clear any previously registered syndrome triggers

-syndrome SYN_NAME
    specify a syndrome by its SYN_NAME
    enclose the name in quotation marks if it includes spaces
    example:
        -syndrome "gila monster bite"

-synclass SYN_CLASS
    any syndrome with the specified SYN_CLASS will act as a trigger
    enclose in quotation marks if it includes spaces
    example:
        -synclass VAMPCURSE

-command [ commandStrs ]
    specify the command to be executed after infection
    remember to include a space after/before the square brackets!
    the following may be added to appropriate commands where relevant:
        \\UNIT_ID
            inserts the ID of the infected unit
        \\LOCATION
            inserts the x, y, z coordinates of the infected unit
        \\SYNDROME_ID
            inserts the ID of the syndrome
    note that:
        \\anything -> \anything
        anything -> anything
    examples:
```

(continues on next page)

(continued from previous page)

```
        -command [ full-heal -unit \\UNIT_ID ]
            heals units when they acquire the specified syndrome
        -command [ modtools/spawn-flow -flowType Dragonfire -location [ \\LOCATION ] ]
            spawns dragonfire at the location of infected units
```

6.5.30 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.

Arguments:

```
-clear
    clear records of normal races
-unit id
    set the target unit
-duration ticks
    how long it should last, or "forever"
-setPrevRace
    make a record of the previous race so that you can
    change it back with -untransform
-keepInventory
    move items back into inventory after transformation
-race raceName
-caste casteName
-suppressAnnouncement
    don't show the Unit has transformed into a Blah! event
-untransform
    turn the unit back into what it was before
```

CHAPTER 7

User Guides

These pages are detailed guides covering DFHack tools.

7.1 DFHack Example Configuration File Index

The hack/examples folder contains ready-to-use examples of various DFHack configuration files. You can use them by copying them to appropriate folders where DFHack and its plugins can find them (details below). You can use them unmodified, or you can customize them to better suit your preferences.

7.1.1 The `init/` subfolder

The init/ subfolder contains useful DFHack *Init Files* that you can copy into your main Dwarf Fortress folder – the same directory as `dfhack.init`.

onMapLoad_dreamfort.init

This is the config file that comes with the *Dreamfort* set of blueprints, but it is useful (and customizable) for any fort. It includes the following config:

- Calls *ban-cooking* for items that have important alternate uses and should not be cooked. This configuration is only set when a fortress is first started, so later manual changes will not be overridden.
- Automates calling of various fort maintenance and *Bugfixing Scripts*, like *cleanowned* and *fix/stuckdoors*.
- Keeps your manager orders intelligently ordered with *orders* `sort` so no orders block other orders from ever getting completed.
- Periodically enqueues orders to shear and milk shearable and milkable pets.
- Sets up *autofarm* to grow 30 units of every crop, except for pig tails, which is set to 150 units to support the textile industry.
- Sets up *seedwatch* to keep 30 of every type of seed.

- Configures *prioritize* to automatically boost the priority of important and time-sensitive tasks that could otherwise get ignored in busy forts, like hauling food, tanning hides, storing items in vehicles, pulling levers, and removing constructions.
- Optimizes *autobutcher* settings for raising geese, alpacas, sheep, llamas, and pigs. Adds sensible defaults for all other animals, including dogs and cats. There are instructions in the file for customizing the settings for other combinations of animals. These settings are also only set when a fortress is first started, so any later changes you make to autobutcher settings won't be overridden.
- Enables *automelt*, *tailor*, *zone*, *nestboxes*, and *autonestbox*.

7.1.2 The `orders/` subfolder

The orders/ subfolder contains manager orders that, along with the `onMapLoad_dreamfort.init` file above, allow a fort to be self-sustaining. Copy them to your `dfhack-config/orders/` folder and import as required with the *orders* DFHack plugin.

basic.json

This collection of orders handles basic fort necessities:

- prepared meals and food products (and by-products like oil)
- booze/mead
- thread/cloth/dye
- pots/jugs/buckets/mugs
- bags of leather, cloth, silk, and yarn
- crafts and totems from otherwise unusable by-products
- mechanisms/cages
- splints/crutches
- lye/soap
- ash/potash
- beds/wheelbarrows/minecarts
- scrolls

You should import it as soon as you have enough dwarves to perform the tasks. Right after the first migration wave is usually a good time.

furnace.json

This collection creates basic items that require heat. It is separated out from `basic.json` to give players the opportunity to set up magma furnaces first in order to save resources. It handles:

- charcoal (including smelting of bituminous coal and lignite)
- pearlash
- sand
- green/clear/crystal glass
- adamantine processing

- item melting

Orders are missing for plaster powder until DF bug 11803 is fixed.

military.json

This collection adds high-volume smelting jobs for military-grade metal ores and produces weapons and armor:

- leather backpacks/waterskins/cloaks/quivers/armor
- bone/wooden bolts
- smelting for platinum, silver, steel, bronze, bismuth bronze, and copper (and their dependencies)
- bronze/bismuth bronze/copper bolts
- platinum/silver/steel/iron/bismuth bronze/bronze/copper weapons and armor, with checks to ensure only the best available materials are being used

If you set a stockpile to take weapons and armor of less than masterwork quality and turn on *automelt* (like what *Dreamfort* provides on its industry level), these orders will automatically upgrade your military equipment to masterwork. Make sure you have a lot of fuel (or magma forges and furnaces) before you turn `automelt` on, though!

This file should only be imported, of course, if you need to equip a military.

smelting.json

This collection adds smelting jobs for all ores. It includes handling the ores already managed by `military.json`, but has lower limits. This ensures all ores will be covered if a player imports `smelting` but not `military`, but the higher-volume `military` orders will take priority if both are imported.

rockstock.json

This collection of orders keeps a small stock of all types of rock furniture. This allows you to do ad-hoc furnishings of guildhalls, libraries, temples, or other rooms with *buildingplan* and your masons will make sure there is always stock on hand to fulfill the plans.

glassstock.json

Similar to `rockstock` above, this collection keeps a small stock of all types of glass furniture. If you have a functioning glass industry, this is more sustainable than `rockstock` since you can never run out of sand. If you have plenty of rock and just want the variety, you can import both `rockstock` and `glassstock` to get a mixture of rock and glass furnishings in your fort.

There are a few items that `glassstock` produces that `rockstock` does not, since there are some items that can not be made out of rock, for example:

- tubes and corkscrews for building magma-safe screw pumps
- windows
- terrariums (as an alternative to wooden cages)

7.1.3 The `professions/` subfolder

The professions/ subfolder contains professions, or sets of related labors, that you can assign to your dwarves with the DFHack *manipulator* plugin. Copy them into the `professions/` subdirectory under the main Dwarf Fortress folder (you may have to create this subdirectory) and assign them to your dwarves in the manipulator UI, accessible from the `units` screen via the `l` hotkey. Make sure that the `manipulator` plugin is enabled in your `dfhack.init` file! You can assign a profession to a dwarf by selecting the dwarf in the `manipulator` UI and hitting `p`. The list of professions that you copied into the `professions/` folder will show up for you to choose from. This is very useful for assigning roles to new migrants to ensure that all the tasks in your fort have adequate numbers of dwarves attending to them.

If you'd rather use Dwarf Therapist to manage your labors, it is easy to import these professions to DT and use them there. Simply assign the professions you want to import to a dwarf. Once you have assigned a profession to at least one dwarf, you can select "Import Professions from DF" in the DT "File" menu. The professions will then be available for use in DT.

In the charts below, the "At Start" and "Max" columns indicate the approximate number of dwarves of each profession that you are likely to need at the start of the game and how many you are likely to need in a mature fort.

Profession	At Start	Max	Description
Chef	0	3	Buchery, Tanning, and Cooking. It is important to focus just a few dwarves on cooking since well-crafted meals make dwarves very happy. They are also an excellent trade good.
Craftsdwarf	0	4-6	All labors used at Craftsdwarf's workshops, Glassmaker's workshops, and kilns.
Doctor	0	2-4	The full suite of medical labors, plus Animal Caretaking for those using the dwarfvet plugin.
Farmer	1	4	Food- and animal product-related labors. This profession also has the `Alchemist` labor enabled since they need to focus on food-related jobs, though you might want to disable `Alchemist` for your first farmer until there are actual farming duties to perform.
Fisherdwarf	0	0-1	Fishing and fish cleaning. If you assign this profession to any dwarf, be prepared to be inundated with fish. Fisherdwarves *never stop fishing*. Be sure to also run `prioritize -a PrepareRawFish ExtractFromRawFish` (or use the `onMapLoad_dreamfort.init` file above) or else caught fish will just be left to rot.
Hauler	0	>20	All hauling labors plus Siege Operating, Mechanic (so haulers can assist in reloading traps) and Architecture (so haulers can help build massive windmill farms and pump stacks). As you accumulate enough Haulers, you can turn off hauling labors for other dwarves so they can focus on their skilled tasks. You may also want to restrict your Mechanic's workshops to only skilled mechanics so your haulers don't make low-quality mechanisms.
Laborer	0	10-12	All labors that don't improve quality with skill, such as Soapmaking and furnace labors.
Marksdwarf	0	10-30	Similar to Hauler. See the description for Meleedwarf below for more details.
Mason	2	2-4	Masonry, Gem Cutting/Encrusting, and Architecture. In the early game, you may need to run "*prioritize* ConstructBuilding" to get your masons to build wells and bridges if they are too busy crafting stone furniture.
Meleedwarf	0	20-50	Similar to Hauler, but without most civilian labors. This profession is separate from Hauler so you can find your military dwarves easily. Meleedwarves and Marksdwarves have Mechanics and hauling labors enabled so you can temporarily deactivate your military after sieges and allow your military dwarves to help clean up.
Migrant	0	0	You can assign this profession to new migrants temporarily while you sort them into professions. Like Marksdwarf and Meleedwarf, the purpose of this profession is so you can find your new dwarves more easily.
Miner	2	2-10	Mining and Engraving. This profession also has the `Alchemist` labor enabled, which disables hauling for those using the *autohauler* plugin. Once the need for Miners tapers off in the late game, dwarves with this profession make good military dwarves, wielding their picks as weapons.
Outdoorsdwarf	1	2-4	Carpentry, Bowyery, Woodcutting, Animal Training, Trapping, Plant Gathering, Beekeeping, and Siege Engineering.
Smith	0	2-4	Smithing labors. You may want to specialize your Smiths to focus on a single smithing skill to maximize equipment quality.
StartManager	1	0	All skills not covered by the other starting professions (Miner, Mason, Outdoorsdwarf, and Farmer), plus a few overlapping skills to assist in critical tasks at the beginning of the game. Individual labors should be turned off as migrants are assigned more specialized professions that cover them, and the StartManager dwarf can eventually convert to some other profession.
Tailor	0	2	Textile industry labors: Dying, Leatherworking, Weaving, and Clothesmaking.

A note on autohauler

These profession definitions are designed to work well with or without the *autohauler* plugin (which helps to keep your dwarves focused on skilled labors instead of constantly being distracted by hauling). If you do want to use autohauler, adding the following lines to your `onMapLoad.init` file will configure it to let the professions manage the "Feed water to civilians" and "Recover wounded" labors instead of enabling those labors for all hauling dwarves:

```
on-new-fortress enable autohauler
on-new-fortress autohauler FEED_WATER_CIVILIANS allow
on-new-fortress autohauler RECOVER_WOUNDED allow
```

7.2 Quickfort Keystroke Alias Guide

Aliases allow you to use simple words to represent complicated key sequences when configuring buildings and stockpiles in quickfort `#query` blueprints.

For example, say you have the following `#build` and `#place` blueprints:

```
#build masonry workshop
~, ~,~,`,`,`
~,wm,~,`,`,`
~, ~,~,`,`,`

#place stockpile for mason
~,~,~,s,s,s
~,~,~,s,s,s
~,~,~,s,s,s
```

and you want to configure the stockpile to hold only non-economic ("other") stone and to give to the adjacent mason workshop. You could write the key sequences directly:

```
#query configure stockpile with expanded key sequences
~,~,~,s{Down 5}deb{Right}{Down 2}p^,`,`
~,~,~,g{Left 2}&,                   `,`
~,~,~,`,                            `,`
```

or you could use aliases:

```
#query configure stockpile with aliases
~,~,~,otherstone,`,`
~,~,~,give2left, `,`
~,~,~,`,         `,`
```

If the stockpile had only a single tile, you could also replay both aliases in a single cell:

```
#query configure mason with multiple aliases in one cell
~,~,~,{otherstone}{give2left},`,`
~,~,~,`,                      `,`
~,~,~,`,                      `,`
```

With aliases, blueprints are much easier to read and understand. They also save you from having to copy the same long key sequences everywhere.

7.2.1 Alias definition files

DFHack comes with a library of aliases for you to use that are always available when you run a `#query` blueprint. Many blueprints can be built with just those aliases. This "standard alias library" is stored in data/quickfort/aliases-common.txt (installed under the `hack` folder in your DFHack installation). The aliases in that file are described at the *bottom of this document*.

Please do not edit the aliases in the standard library directly. The file will get overwritten when DFHack is updated and you'll lose your changes. Instead, add your custom aliases to dfhack-config/quickfort/aliases.txt or directly to your blueprints in an *#aliases* section. Your custom alias definitions take precedence over any definitions in the standard library.

7.2.2 Alias syntax and usage

The syntax for defining aliases is:

```
aliasname: expansion
```

Where `aliasname` is at least two letters or digits long (dashes and underscores are also allowed) and `expansion` is whatever you would type into the DF UI.

You use an alias by typing its name into a `#query` blueprint cell where you want it to be applied. You can use an alias by itself or as part of a larger sequence, potentially with other aliases. If the alias is the only text in the cell, the alias name is matched and its expansion is used. If the alias has other keys before or after it, the alias name must be surrounded in curly brackets (`{` and `}`). An alias can be surrounded in curly brackets even if it is the only text in the cell, it just isn't necesary. For example, the following blueprint uses the `aliasname` alias by itself in the first two rows and uses it as part of a longer sequence in the third row:

```
#query apply alias 'aliasname' in three different ways
aliasname
{aliasname}
literaltext{aliasname}literaltext
```

For a more concrete example of an alias definition, a simple alias that configures a stockpile to have no bins (`C`) and no barrels (`E`) assigned to it would look like this:

```
nocontainers: CE
```

The alias definition can also contain references to other aliases by including the alias names in curly brackets. For example, `nocontainers` could be equivalently defined like this:

```
nobins: C
nobarrels: E
nocontainers: {nobins}{nobarrels}
```

Aliases used in alias definitions *must* be surrounded by curly brackets, even if they are the only text in the definition:

```
alias1: text1
alias2: alias1
alias3: {alias1}
```

Here, `alias1` and `alias3` expand to `text1`, but `alias2` expands to the literal text `alias1`.

Keycodes

Non-printable characters, like the arrow keys, are represented by their keycode name and are also surrounded by curly brackets, like `{Right}` or `{Enter}`. Keycodes are used exactly like aliases – they just have special expansions that you wouldn't be able to write yourself. In order to avoid naming conflicts between aliases and keycodes, the convention is to start aliases with a lowercase letter.

Any keycode name from the DF interface definition file (data/init/interface.txt) is valid, but only a few keycodes are actually useful for blueprints:

```
Up
Down
Left
Right
Enter
ESC
Backspace
Space
Tab
```

There is also one pseudo-keycode that quickfort recognizes:

```
Empty
```

which has an empty expansion. It is primarily useful for defining blank default values for *Sub-aliases*.

Repetitions

Anything enclosed within curly brackets can also have a number, indicating how many times that alias or keycode should be repeated. For example: `{togglesequence 9}` or `{Down 5}` will repeat the `togglesequence` alias nine times and the `Down` keycode five times, respectively.

Modifier keys

Ctrl, Alt, and Shift modifiers can be specified for the next key by adding them into the key sequence. For example, Alt-h is written as `{Alt}h`.

Shorthand characters

Some frequently-used keycodes are assigned shorthand characters. Think of them as single-character aliases that don't need to be surrounded in curly brackets:

```
&   expands to {Enter}
@   expands to {Shift}{Enter}
~   expands to {Alt}
!   expands to {Ctrl}
^   expands to {ESC}
```

If you need literal versions of the shorthand characters, surround them in curly brackets, for example: use `{!}` for a literal exclamation point.

Built-in aliases

Most aliases that come with DFHack are in `aliases-common.txt`, but there is one alias built into the code for the common shorthand for "make room":

```
r+  expands to r+{Enter}
```

This needs special code support since + can't normally be used in alias names. You can use it just like any other alias, either by itself in a cell (`r+`) or surrounded in curly brackets (`{r+}`).

Sub-aliases

You can specify sub-aliases that will only be defined while the current alias is being resolved. This is useful for "injecting" custom behavior into the middle of a larger alias. As a simple example, the `givename` alias is defined like this:

```
givename: !n{name}&
```

Note the use of the `name` alias inside of the `givename` expansion. In your `#query` blueprint, you could write something like this, say, while over your main drawbridge:

```
{givename name="Front Gate"}
```

The value that you give the sub-alias `name` will be used when the `givename` alias is expanded. Without sub-aliases, we'd have to define `givename` like this:

```
givenameprefix: !n
givenamesuffix: &
```

and use it like this:

```
{givenameprefix}Front Gate{givenamesuffix}
```

which is more difficult to write and more difficult to understand.

A handy technique is to define an alias with some sort of default behavior and then use sub-aliases to override that behavior as necessary. For example, here is a simplified version of the standard `quantum` alias that sets up quantum stockpiles:

```
quantum_enable: {enableanimals}{enablefood}{enablefurniture}...
quantum: {linksonly}{nocontainers}{quantum_enable}
```

You can use the default behavior of `quantum_enable` by just using the `quantum` alias by itself. But you can override `quantum_enable` to just enable furniture for some specific stockpile like this:

```
{quantum quantum_enable={enablefurniture}}
```

If an alias uses a sub-alias in its expansion, but the sub-alias is not defined when the alias is used, quickfort will halt the `#query` blueprint with an error. If you want your aliases to work regardless of whether sub-aliases are defined, then you must define them with default values like `quantum_enable` above. If a default value should be blank, like the `name` sub-alias used by the `givename` alias above, define it with the `{Empty}` pesudo-keycode:

```
name: {Empty}
```

Sub-aliases must be in one of the following formats:

```
subaliasname=keyswithnospaces
subaliasname="keys with spaces or {aliases}"
subaliasname={singlealias}
```

If you specify both a sub-alias and a number of repetitions, the number for repetitions goes last, right before the `}`:

```
{alias subaliasname=value repetitions}
```

7.2.3 Beyond query mode

`#query` blueprints normally do things in DF `q`uery mode, but nobody said that we have to *stay* in query mode. `#query` blueprints send arbitrary key sequences to Dwarf Fortress. Anything you can do by typing keys into DF, you can do in a `#query` blueprint. It is absolutely fine to temporarily exit out of query mode, go into, say, hauling or zone or hotkey mode, and do whatever needs to be done.

You just have to make certain to exit out of that alternate mode and get back into `q`uery mode at the end of the key sequence. That way quickfort can continue on configuring the next tile – a tile configuration that assumes the game is still in query mode.

For example, here is the standard library alias for giving a name to a zone:

```
namezone: ^i{givename}^q
```

The first `^` exits out of query mode. Then `i` enters zones mode. We then reuse the standard alias for giving something a name. Finally, we exit out of zones mode with another `^` and return to `q`uery mode.

7.2.4 The DFHack standard alias library

DFHack comes with many useful aliases for you to use in your blueprints. Many blueprints can be built with just these aliases alone, with no custom aliases required.

This section goes through all aliases provided by the DFHack standard alias library, discussing their intended usage and detailing sub-aliases that you can define to customize their behavior.

If you do define your own custom aliases in `dfhack-config/quickfort/aliases.txt`, try to build on library alias components. For example, if you create an alias to modify particular furniture stockpile settings, start your alias with `{furnitureprefix}` instead of `s{Down 2}`. Using library prefixes will allow library sub-aliases to work with your aliases just like they do with library aliases. In this case, using `{furnitureprefix}` will allow your stockpile customization alias to work with both stockpiles and hauling routes.

Note that some aliases use the DFHack-provided search prompts. If you get errors while running `#query` blueprints, ensure the DFHack *search* plugin is enabled.

Naming aliases

These aliases give descriptive names to workshops, levers, stockpiles, zones, etc. Dwarf Fortress building, stockpile, and zone names have a maximum length of 20 characters.

Alias	Sub-aliases
givename	name
namezone	name

`givename` works anywhere you can hit Ctrl-n to customize a name, like when the cursor is over buildings and stockpiles. Example:

```
#place
f(10x2)

#query
{booze}{givename name=booze}
```

`namezone` is intended to be used when over an activity zone. It includes commands to get into zones mode, set the zone name, and get back to query mode. Example:

```
#zone
n(2x2)

#query
{namezone name="guard dog pen"}
```

Quantum stockpile aliases

These aliases make it easy to create minecart stop-based quantum stockpiles.

Alias	Sub-aliases
quantum	name quantum_enable
quantumstopfromnorth	name stop_name route_enable
quantumstopfromsouth	
quantumstopfromeast	
quantumstopfromwest	
sp_link	move move_back
quantumstop	name stop_name route_enable move move_back sp_links

The idea is to use a minecart on a track stop to dump an infinite number of items into a receiving "quantum" stockpile, which significantly simplifies stockpile management. These aliases configure the quantum stockpile and hauling route that make it all work. Here is a complete example for quantum stockpiling weapons, armor, and ammunition. It has a 3x1 feeder stockpile on the bottom (South), the trackstop in the center, and the quantum stockpile on the top (North). Note that the feeder stockpile is the only stockpile that needs to be configured to control which types of items end up in the quantum stockpile. By default, the hauling route and quantum stockpile itself simply accept whatever is put into them.

```
#place
 ,c
 ,
pdz(3x1)

#build
 ,
 ,trackstopN

#query message(remember to assign a minecart to the new route)
 ,quantum
 ,quantumstopfromsouth
nocontainers
```

The `quantum` alias configures a 1x1 stockpile to be a quantum stockpile. It bans all containers and prevents the stockpile from being manually filled. By default, it also enables storage of all item categories (except corpses and refuse), so it doesn't really matter what letter you use to place the stockpile. Refuse is excluded by default since otherwise clothes and armor in the quantum stockpile would rot away. If you want corpses or bones in your quantum stockpile, use `y` and/or `r` to place the stockpile and the `quantum` alias will just enable the remaining types. If you *do* enable refuse in your quantum stockpile, be sure you avoid putting useful clothes or armor in there!

The `quantumstopfromsouth` alias is run over the track stop and configures the hauling route, again, allowing all item categories into the minecart by default so any item that can go into the feeder stockpile can then be placed in the minecart. It also links the hauling route with the feeder stockpile to the South.The track stop does not need to be fully constructed before the `#query` blueprint is run, but the feeder stockpile needs to exist so we can link to it. This means that the three blueprints above can be run one right after another, without any dwarven labor in between them, and the quantum stockpile will work properly.

Finally, the `nocontainers` alias simply configures the feeder stockpile to not have any containers (which would just get in the way here). If we wanted to be more specific about what item types we want in the quantum stockpile, we could configure the feeder stockpile further, for example with standard *stockpile adjustment aliases*.

After the blueprints are run, the last step is to manually assign a minecart to the newly-defined hauling route.

You can define sub-aliases to customize how these aliases work, for example to have fine-grained control over what item types are enabled for the route and quantum stockpile. We'll go over those options below, but first, here is an example for how to just give names to everything:

```
#query message(remember to assign a minecart to the new route)
 ,{quantum name="armory quantum"}
 ,{quantumstopfromsouth name="Armory quantum" stop_name="Armory quantum stop"}
↪{givename name="armory dumper"}
{givename name="armory feeder"}
```

All `name` sub-aliases are completely optional, of course. Keep in mind that hauling route names have a maximum length of 22 characters, hauling route stop names have a maximum length of 21 characters, and all other names have a maximum length of 20 characters.

If you want to be absolutely certain that nothing ends up in your quantum stockpile other than what you've configured in the feeder stockpile, you can set the `quantum_enable` sub-alias for the `quantum` alias. This can prevent, for example, somebody's knocked-out tooth from being considered part of your furniture quantum stockpile when it happened to land on it during a fistfight:

```
#query
{quantum name="furniture quantum" quantum_enable={enablefurniture}}
```

You can have similar control over the hauling route if you need to be more selective about what item types are allowed into the minecart. If you have multiple specialized quantum stockpiles that use a common feeder pile, for example,

you can set the `route_enable` sub-alias:

```
#query
{quantumstopfromsouth name="Steel bar quantum" route_enable="{enablebars}{steelbars}"}
```

Any of the *stockpile configuration aliases* can be used for either the `quantum_enable` or `route_enable` sub-aliases. Experienced Dwarf Fortress players may be wondering how the same aliases can work in both contexts since the keys for entering the configuration screen differ. Fear not! There is some sub-alias magic at work here. If you define your own stockpile configuraiton aliases, you can use the magic yourself by building your aliases on the `*prefix` aliases described later in this guide.

Finally, the `quantumstop` alias is a more general version of the simpler `quantumstopfrom*` aliases. The `quantumstopfrom*` aliases assume that a single feeder stockpile is orthogonally adjacent to your track stop (which is how most people set them up). If your feeder stockpile is somewhere further away, or you have multiple feeder stockpiles to link, you can use the `quantumstop` alias directly. In addition to the sub-aliases used in the `quantumstopfrom*` alias, you can define the `move` and `move_back` sub-aliases, which let you specify the cursor keys required to move from the track stop to the (single) feeder stockpile and back again, respectively:

```
#query
{quantumstop move="{Right 2}{Up}" move_back="{Down}{Left 2}"}
```

If you have multiple stockpiles to link, define the `sp_links` sub-alias, which can chain several `sp_link` aliases together, each with their own movement configuration:

```
#query
{quantumstop sp_links="{sp_link move=""{Right}{Up}"" move_back=""{Down}{Left}""}{sp_
↪link move=""{Right}{Down}"" move_back=""{Up}{Left}""}"}
```

Note the doubled quotes for quoted elements that are within the outer quotes.

Farm plots

Sets a farm plot to grow the first or last type of seed in the list of available seeds for all four seasons. The last seed is usually Plump helmet spawn, suitable for post-embark. But if you only have one seed type, that'll be grown instead.

Alias
growlastcropall
growfirstcropall

Instead of these aliases, though, it might be more useful to use the DFHack *autofarm* plugin.

Stockpile configuration utility aliases

Alias	Sub-aliases
linksonly	
maxbins	
maxbarrels	
nobins	
nobarrels	
nocontainers	
give2up	
give2down	
give2left	
give2right	
give10up	
give10down	
give10left	
give10right	
give	move
togglesequence	
togglesequence2	
masterworkonly	prefix
artifactonly	prefix
togglemasterwork	prefix
toggleartifact	prefix

`linksonly`, `maxbins`, `maxbarrels`, `nobins`, `nobarrels`, and `nocontainers` set the named basic properties on stockpiles. `nocontainers` sets bins and barrels to 0, but does not affect wheelbarrows since the hotkeys for changing the number of wheelbarrows depend on whether you have DFHack's `tweak max-wheelbarrow` enabled. It is better to set the number of wheelbarrows via the *quickfort* `stockpiles_max_wheelbarrows` setting (set to `0` by default), or explicitly when you define the stockpile in the `#place` blueprint.

The `give*` aliases set a stockpile to give to a workshop or another stockpile located at the indicated number of tiles in the indicated direction from the current tile. For example, here we use the `give2down` alias to connect an `otherstone` stockpile with a mason workshop:

```
#place
s,s,s,s,s
s, , , ,s
s, , , ,s
s, , , ,s
s,s,s,s,s

#build
`,`,`,`,`
`, , , ,`
`, ,wm,,`
`, , , ,`
`,`,`,`,`

#query
 , ,give2down
otherstone
```

and here is a generic stone stockpile that gives to a stockpile that only takes flux:

```
#place
s(10x1)
s(10x10)

#query
flux
,
give2up
```

If you want to give to some other tile that is not already covered by the `give2*` or `give10*` aliases, you can use the generic `give` alias and specify the movement keys yourself in the `move` sub-alias. Here is how to give to a stockpile or workshop one z-level above, 9 tiles to the left, and 14 tiles down:

```
#query
{give move="<{Left 9}{Down 14}"}
```

`togglesequence` and `togglesequence2` send `{Down}{Enter}` or `{Down 2}{Enter}` to toggle adjacent (or alternating) items in a list. This is useful when toggling a bunch of related item types in the stockpile config. For example, the `dye` alias in the standard alias library needs to select four adjacent items:

```
dye: {foodprefix}b{Right}{Down 11}{Right}{Down 28}{togglesequence 4}^
```

Finally, the `masterwork` and `artifact` group of aliases configure the corresponding allowable core quality for the stockpile categories that have them. This alias is used to implement category-specific aliases below, like `artifactweapons` and `forbidartifactweapons`.

Stockpile adjustment aliases

For each stockpile item category, there are three standard aliases:

- `*prefix` aliases enter the stockpile configuration screen and position the cursor at a particular item category in the left-most column, ready for further keys that configure the elements within that category. All other stockpile adjustment aliases are built on these prefixes. You can use them yourself to create stockpile adjustment aliases that aren't already covered by the standard library aliases. Using the library prefix instead of creating your own also allows your stockpile configuration aliases to be used for both stockpiles and hauling routes. For example, here is the library definition for `booze`:

  ```
  booze: {foodprefix}b{Right}{Down 5}p{Down}p^
  ```

- `enable*` aliases enter the stockpile configuration screen, enable all subtypes of the named category, and exit the stockpile configuration screen
- `disable*` aliases enter the stockpile configuration screen, disable all subtypes of the named category, and exit the stockpile configuration screen

Prefix	Enable	Disable
animalsprefix	enableanimals	disableanimals
foodprefix	enablefood	disablefood
furnitureprefix	enablefurniture	disablefurniture
corpsesprefix	enablecorpses	disablecorpses
refuseprefix	enablerefuse	disablerefuse
stoneprefix	enablestone	disablestone
ammoprefix	enableammo	disableammo
coinsprefix	enablecoins	disablecoins
barsprefix	enablebars	disablebars
gemsprefix	enablegems	disablegems
finishedgoodsprefix	enablefinishedgoods	disablefinishedgoods
leatherprefix	enableleather	disableleather
clothprefix	enablecloth	disablecloth
woodprefix	enablewood	disablewood
weaponsprefix	enableweapons	disableweapons
armorprefix	enablearmor	disablearmor
sheetprefix	enablesheet	disablesheet

Then, for each item category, there are aliases that manipulate interesting subsets of that category:

- Exclusive aliases forbid everthing within a category and then enable only the named item type (or named class of items)
- `forbid*` aliases forbid the named type and leave the rest of the stockpile untouched.
- `permit*` aliases permit the named type and leave the rest of the stockpile untouched.

Note that for specific item types (items in the third stockpile configuration column), you can only toggle the item type on and off. Aliases can't know whether sending the `{Enter}` key will enable or disable the type. The `forbid*` aliases that affect these item types assume the item type was enabled and toggle it off. Likewise, the `permit*` aliases assume the item type was disabled and toggle it on. If the item type is not in the expected enabled/disabled state when the alias is run, the aliases will not behave properly.

Animal stockpile adjustments

Exclusive	Forbid	Permit
cages	forbidcages	permitcages
traps	forbidtraps	permittraps

Food stockpile adjustments

Exclusive	Forbid	Permit
preparedfood	forbidpreparedfood	permitpreparedfood
unpreparedfish	forbidunpreparedfish	permitunpreparedfish
plants	forbidplants	permitplants
booze	forbidbooze	permitbooze
seeds	forbidseeds	permitseeds
dye	forbiddye	permitdye
tallow	forbidtallow	permittallow
miscliquid	forbidmiscliquid	permitmiscliquid
wax	forbidwax	permitwax

Furniture stockpile adjustments

Exclusive	Forbid	Permit
pots	forbidpots	permitpots
bags		
buckets	forbidbuckets	permitbuckets
sand	forbidsand	permitsand
masterworkfurniture	forbidmasterworkfurniture	permitmasterworkfurniture
artifactfurniture	forbidartifactfurniture	permitartifactfurniture

Notes:

- The `bags` alias excludes coffers and other boxes by forbidding all materials other than cloth, yarn, silk, and leather. Therefore, it is difficult to create `forbidbags` and `permitbags` without affecting other types of furniture stored in the same stockpile.
- Because of the limitations of Dwarf Fortress, `bags` cannot distinguish between empty bags and bags filled with gypsum powder.

Refuse stockpile adjustments

Exclusive	Forbid	Permit
corpses	forbidcorpses	permitcorpses
rawhides	forbidrawhides	permitrawhides
tannedhides	forbidtannedhides	permittannedhides
skulls	forbidskulls	permitskulls
bones	forbidbones	permitbones
shells	forbidshells	permitshells
teeth	forbidteeth	permitteeth
horns	forbidhorns	permithorns
hair	forbidhair	permithair
craftrefuse	forbidcraftrefuse	permitcraftrefuse

Notes:

- `craftrefuse` includes everything a craftsdwarf can use: skulls, bones, shells, teeth, horns, and hair.

Stone stockpile adjustments

Exclusive	Forbid	Permit
metal	forbidmetal	permitmetal
iron	forbidiron	permitiron
economic	forbideconomic	permiteconomic
flux	forbidflux	permitflux
plaster	forbidplaster	permitplaster
coalproducing	forbidcoalproducing	permitcoalproducing
otherstone	forbidotherstone	permitotherstone
bauxite	forbidbauxite	permitbauxite
clay	forbidclay	permitclay

Ammo stockpile adjustments

Exclusive	Forbid	Permit
bolts		
	forbidmetalbolts	
	forbidwoodenbolts	
	forbidbonebolts	
masterworkammo	forbidmasterworkammo	permitmasterworkammo
artifactammo	forbidartifactammo	permitartifactammo

Bar stockpile adjustments

Exclusive	Forbid
bars	forbidbars
metalbars	forbidmetalbars
ironbars	forbidironbars
steelbars	forbidsteelbars
pigironbars	forbidpigironbars
otherbars	forbidotherbars
coal	forbidcoal
potash	forbidpotash
ash	forbidash
pearlash	forbidpearlash
soap	forbidsoap
blocks	forbidblocks

Gem stockpile adjustments

Exclusive	Forbid
roughgems	forbidroughgems
roughglass	forbidroughglass
cutgems	forbidcutgems
cutglass	forbidcutglass
cutstone	forbidcutstone

Finished goods stockpile adjustments

Exclusive	Forbid	Permit
jugs		
crafts	forbidcrafts	permitcrafts
goblets	forbidgoblets	permitgoblets
masterworkfinishedgoods	forbidmasterworkfinishedgoods	permitmasterworkfinishedgoods
artifactfinishedgoods	forbidartifactfinishedgoods	permitartifactfinishedgoods

Cloth stockpile adjustments

Exclusive
thread
adamantinethread
cloth
adamantinecloth

Weapon stockpile adjustments

Exclusive	Forbid	Permit
	forbidweapons	permitweapons
	forbidtrapcomponents	permittrapcomponents
metalweapons	forbidmetalweapons	permitmetalweapons
	forbidstoneweapons	permitstoneweapons
	forbidotherweapons	permitotherweapons
ironweapons	forbidironweapons	permitironweapons
bronzeweapons	forbidbronzeweapons	permitbronzeweapons
copperweapons	forbidcopperweapons	permitcopperweapons
steelweapons	forbidsteelweapons	permitsteelweapons
masterworkweapons	forbidmasterworkweapons	permitmasterworkweapons
artifactweapons	forbidartifactweapons	permitartifactweapons

Armor stockpile adjustments

Exclusive	Forbid	Permit
metalarmor	forbidmetalarmor	permitmetalarmor
otherarmor	forbidotherarmor	permitotherarmor
ironarmor	forbidironarmor	permitironarmor
bronzearmor	forbidbronzearmor	permitbronzearmor
copperarmor	forbidcopperarmor	permitcopperarmor
steelarmor	forbidsteelarmor	permitsteelarmor
masterworkarmor	forbidmasterworkarmor	permitmasterworkarmor
artifactarmor	forbidartifactarmor	permitartifactarmor

7.3 Blueprint Library Index

This guide contains a high-level overview of the blueprints available in the quickfort blueprint library. You can list library blueprints by running `quickfort list --library` or by hitting `Altl` in the `quickfort gui` interactive dialog.

Each file is hyperlinked to its online version so you can see exactly what the blueprints do before you run them.

7.3.1 Whole fort blueprint sets

These files contain the plans for entire fortresses. Each file has one or more help sections that walk you through how to build the fort, step by step.

- library/dreamfort.csv
- library/quickfortress.csv

Dreamfort

Dreamfort is a fully functional, self-sustaining fortress with defenses, farming, a complete set of workshops, self-managing quantum stockpiles, a grand dining hall, hospital, jail, fresh water well system, guildhalls, noble suites, and bedrooms for hundreds of dwarves. It also comes with manager work orders to automate basic fort needs, such as food, booze, and item production. It can function by itself or as the core of a larger, more ambitious fortress. Read the high-level walkthrough by running `quickfort run library/dreamfort.csv` and list the walkthroughs for the individual levels by running `quickfort list -l dreamfort -m notes` or `quickfort gui -l dreamfort notes`.

Dreamfort blueprints are available for easy viewing and copying online.

The online spreadsheets also include embark profile suggestions, a complete example embark profile, and a convenient checklist from which you can copy the `quickfort` commands.

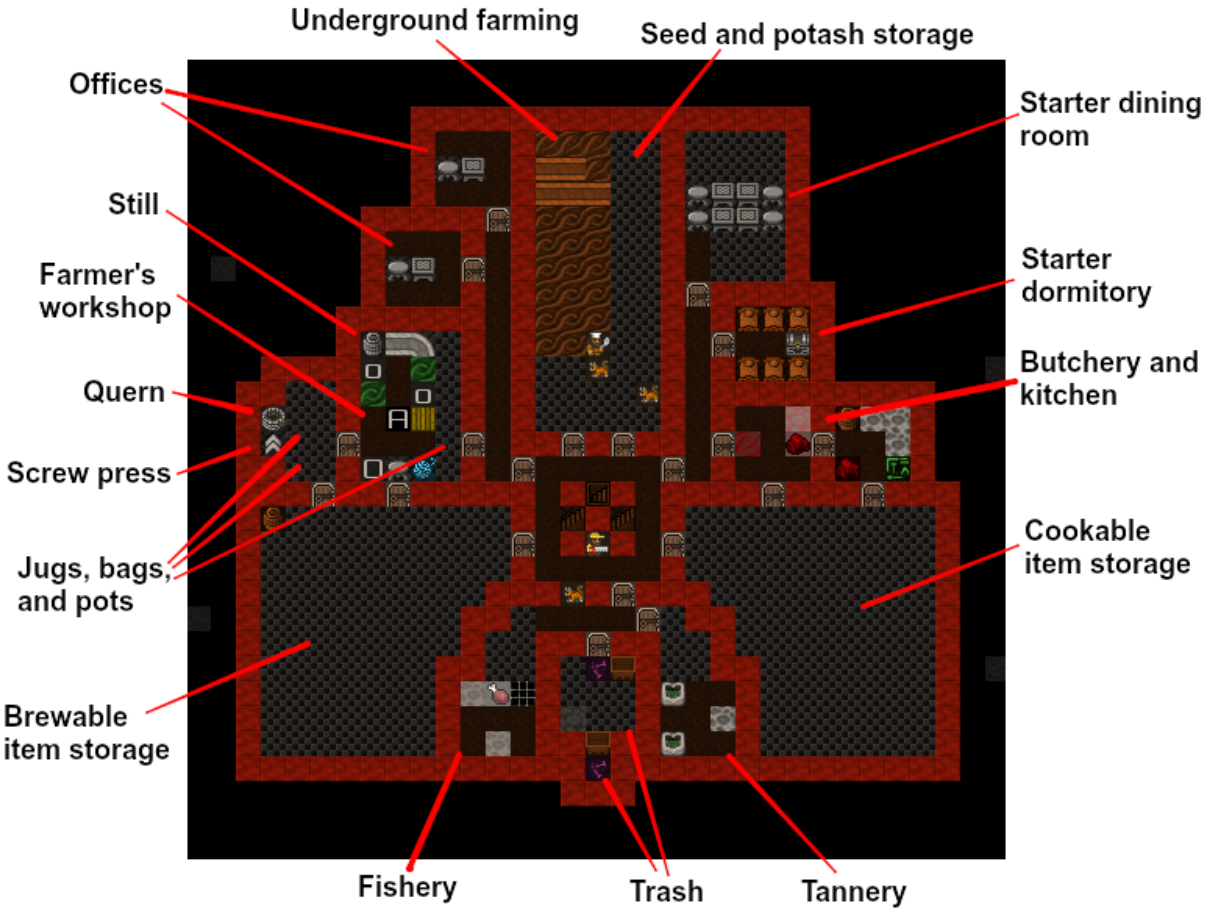
You can download a fully built Dreamfort-based fort from dffd, load it, and explore it interactively.

Visual overview

Here are some annotated screenshots of the major levels (or click here for a slideshow).

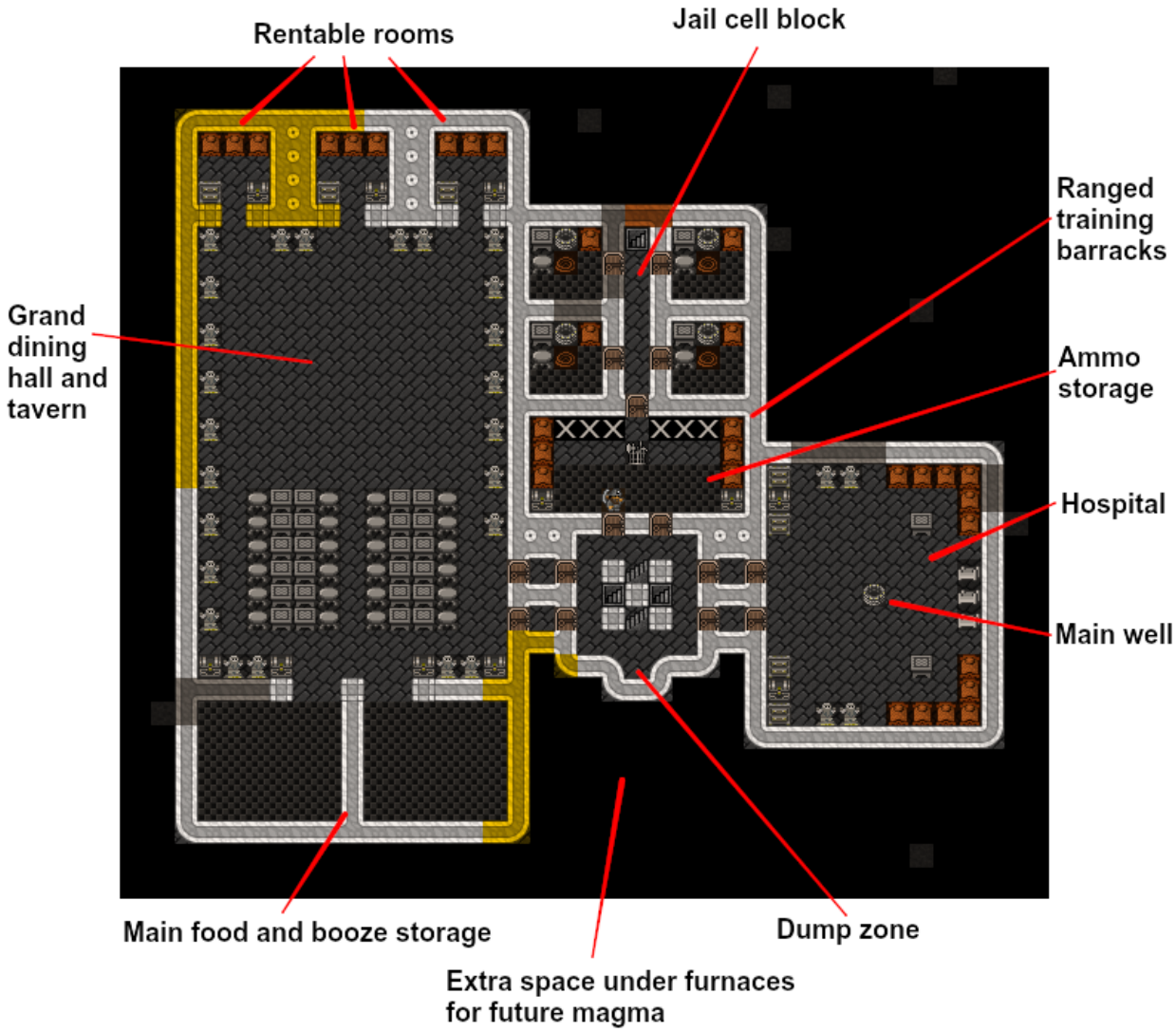
Surface level

Farming level

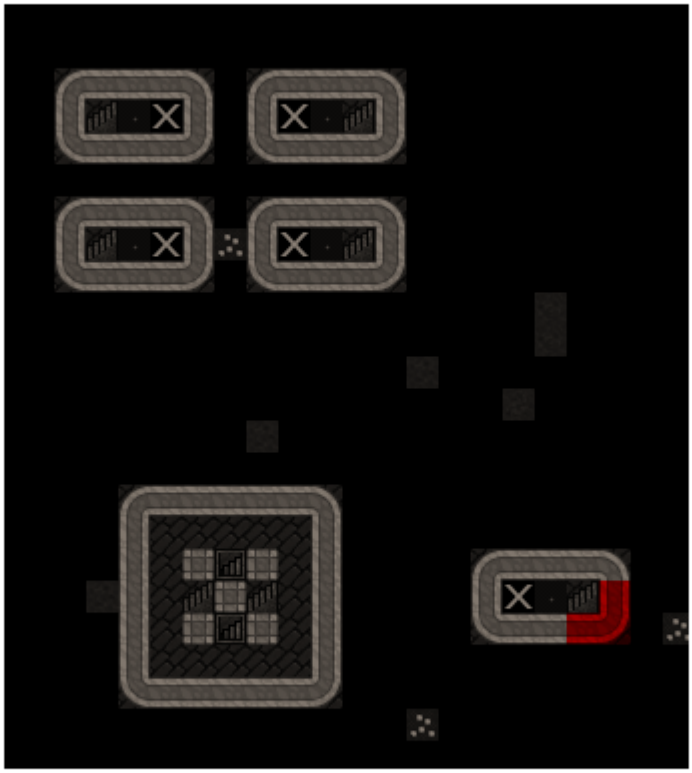


Industry level

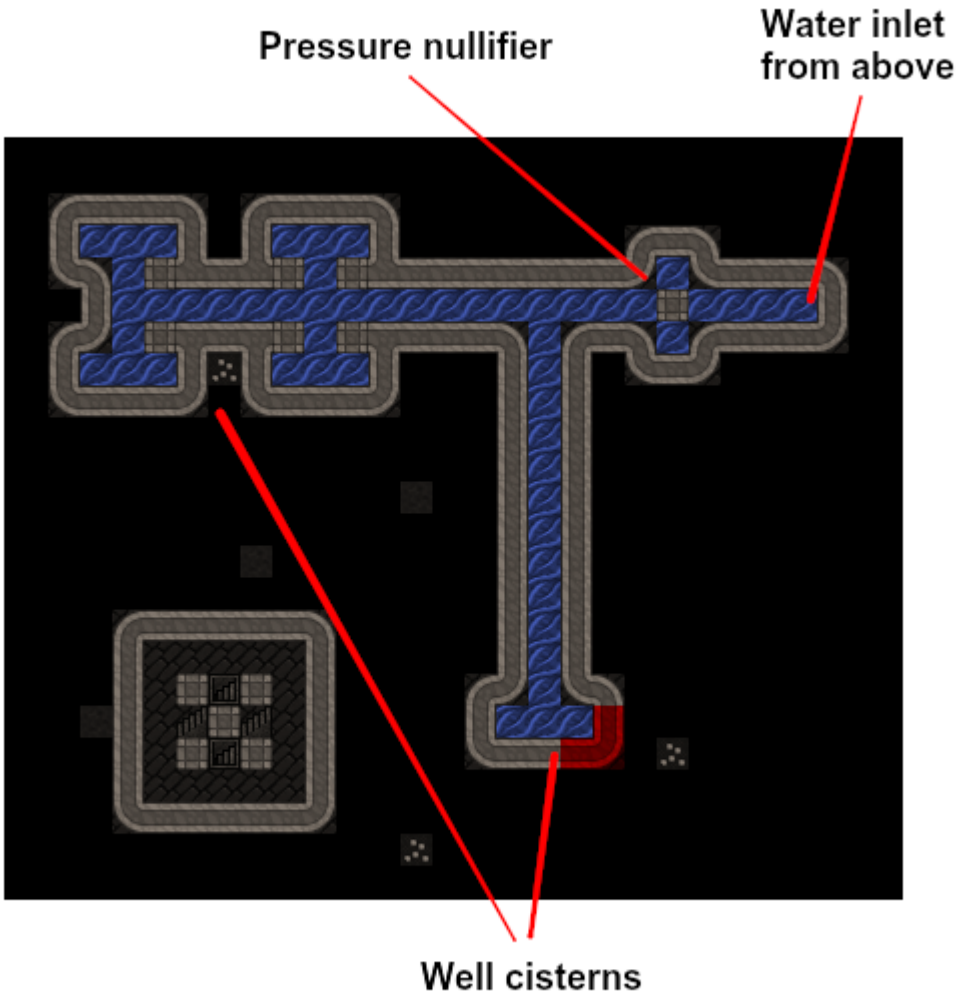
Services level



Fill cisterns with an aqueduct or bucket brigade



Example plumbing to fill cisterns

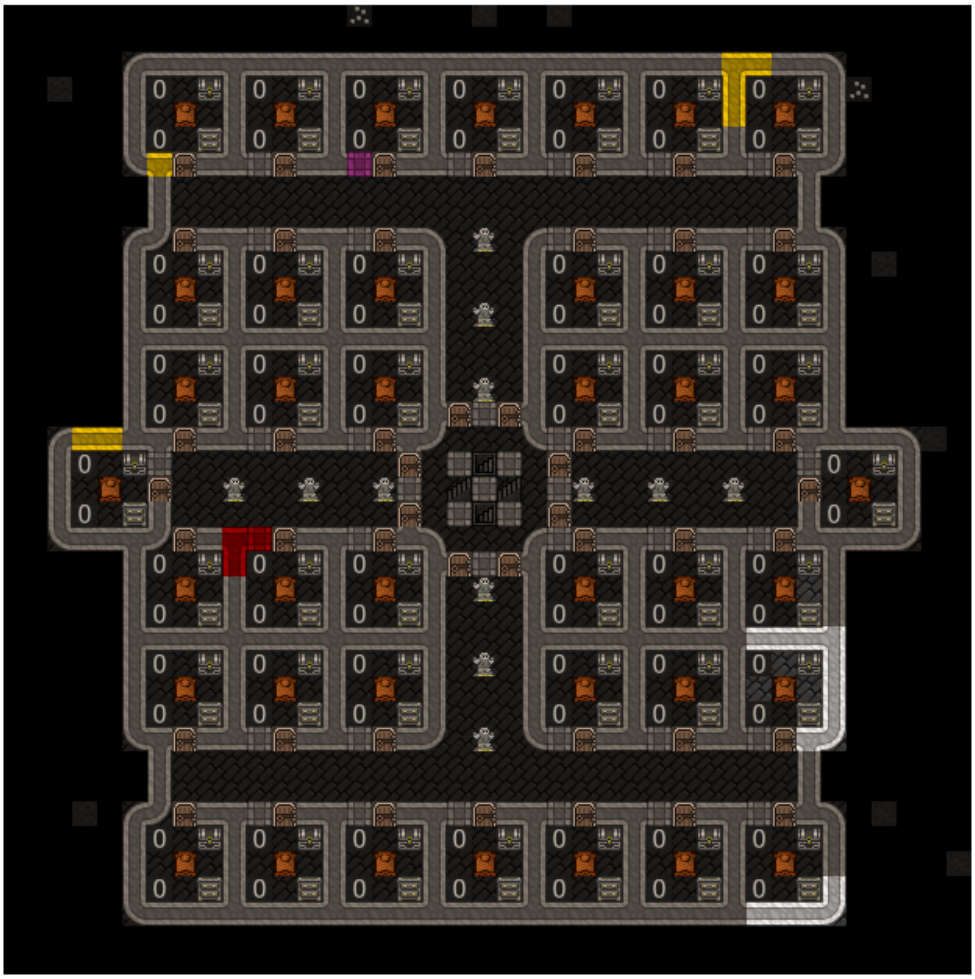


Guildhall level

Noble suites

Apartments

Spacious apartments make happy dwarves!
Doubles as a tomb. 40 rooms/80 urns per level.



The Quick Fortress

The Quick Fortress is an updated version of the example fortress that came with Python Quickfort 2.0 (the utility that inspired DFHack quickfort). While it is not a complete fortress by itself, it is much simpler than Dreamfort and is good for a first introduction to *quickfort* blueprints. Read its walkthrough with `quickfort run library/quickfortress.csv`.

7.3.2 Layout helpers

These files simply draw diagonal marker-mode lines starting from the cursor. They are especially useful for finding the center of the map when you are planning your fortress. Once you are done using them for alignment, use `quickfort undo` at the same cursor position to make them disappear. Since these `#dig` blueprints can only mark undug wall tiles for mining, they are best used underground. They won't do much on the surface, where there aren't many walls.

- library/layout-helpers/mark_up_left.csv
- library/layout-helpers/mark_up_right.csv
- library/layout-helpers/mark_down_right.csv
- library/layout-helpers/mark_down_left.csv

7.3.3 Bedrooms

These are popular bedroom layouts from the Bedroom design page on the wiki. Each file has `#dig`, `#build`, and `#query` blueprints to dig the rooms, build the furniture, and configure the beds as bedrooms, respectively.

- library/bedrooms/48-4-Raynard_Whirlpool_Housing.csv
- library/bedrooms/95-9-Hactar1_3_Branch_Tree.csv
- library/bedrooms/28-3-Modified_Windmill_Villas.csv

7.3.4 Tombs

These blueprints have burial plot layouts for fortress that expect a lot of casualties.

- library/tombs/Mini_Saracen.csv
- library/tombs/The_Saracen_Crypts.csv

7.3.5 Exploratory mining

Several mining patterns to choose from when searching for gems or ores. The patterns can be repeated up or down z-levels (via quickfort's `--repeat` commandline option) for exploring through the depths.

- library/exploratory-mining/tunnels.csv
- library/exploratory-mining/vertical-mineshafts.csv
- library/exploratory-mining/connected-mineshafts.csv

7.3.6 Miscellaneous

Extra blueprints that are useful in specific situations.

- library/aquifer_tap.csv
- library/embark.csv
- library/pump_stack.csv

Light Aquifer Tap

The aquifer tap helps you create a safe, everlasting source of fresh water from a light aquifer. See the step-by-step guide, including informaton on how to create a drainage system so your dwarves don't drown when digging the tap, by running `quickfort run library/aquifer_tap.csv -n /help`.

You can see how to nullify the water pressure (so you don't flood your fort) in the *Dreamfort screenshot above*.

Blueprint spreadsheet also available online

Post-embark

The embark blueprints are useful directly after embark. It contains a `#build` blueprint that builds important starting workshops (mason, carpenter, mechanic, and craftsdwarf) and a `#place` blueprint that lays down a pattern of useful starting stockpiles.

Pump Stack

The pump stack blueprints help you move water and magma up to move convenient locations in your fort. See the step-by-step guide for using it by running `quickfort run library/pump_stack.csv -n /help`.

Blueprint spreadsheet also available online

7.4 Quickfort Blueprint Editing Guide

Quickfort is a DFHack script that helps you build fortresses from "blueprint" .csv and .xlsx files. Many applications exist to edit these files, such as MS Excel and Google Sheets. Most layout and building-oriented DF commands are supported through the use of multiple files or spreadsheets, each describing a different phase of DF construction: designation, building, placing stockpiles/zones, and setting configuration.

The original idea came from Valdemar's auto-designation macro. Joel Thornton reimplemented the core logic in Python and extended its functionality with Quickfort 2.0. This DFHack-native implementation, called "DFHack Quickfort" or just "quickfort", builds upon Quickfort 2.0's formats and features. Any blueprint that worked in Python Quickfort 2.0 should work with DFHack Quickfort. DFHack Quickfort interacts with Dwarf Fortress memory structures directly, allowing for instantaneous blueprint application, error checking and recovery, and many other advanced features.

This guide focuses on DFHack Quickfort's capabilities and teaches players how to understand and create blueprint files. Some of the text was originally written by Joel Thornton, reused here with his permission.

For those just looking to apply existing blueprints, check out the *quickfort command's documentation* for syntax. There are also many ready-to-use blueprints available in the `blueprints/library` subfolder in your DFHack installation. Browse them on your computer or online, or run `quickfort list -l` at the `[DFHack]#` prompt to list them, and then `quickfort run` to apply them to your fort!

Before you become an expert at writing blueprints, though, you should know that the easiest way to make a quickfort blueprint is to build your plan "for real" in Dwarf Fortress and then export your map using the DFHack *blueprint* plugin. You can apply those blueprints as-is in your next fort, or you can fine-tune them with additional features from this guide.

See the *Links* section for more information and online resources.

Table of Contents

7.4.1 Features

- General
  - Manages blueprints to handle all phases of DF construction
  - Supports .csv and multi-worksheet .xlsx blueprint files
  - Near-instant application, even for very large and complex blueprints
  - Blueprints can span multiple z-levels

  - You can package all blueprints and keystroke aliases needed for an entire fortress in a single file for easy sharing
  - "meta" blueprints that simplify the application of sequences of blueprints
  - Undo functionality for dig, build, place, and zone blueprints
  - Rotate blueprints or flip them around to your preference when you apply them to the map
  - Automatic cropping of blueprints so you don't get errors if the blueprint extends off the map
  - Can generate manager orders for everything required by a build blueprint
  - Includes a library of ready-to-use blueprints
  - Blueprint debugging features
- Dig mode
  - Supports all types of designations, including dumping/forbidding items and setting traffic settings
  - Supports setting dig priorities
  - Supports applying dig blueprints in marker mode
  - Handles carving arbitrarily complex minecart tracks, including tracks that cross other tracks
- Build mode
  - Fully integrated with DFHack buildingplan: you can place buildings before manufacturing building materials and you can use the buildingplan UI for setting materials preferences
  - Designate entire constructions in mid-air without having to wait for each tile to become supported
  - Automatic expansion of building footprints to their minimum dimensions, so only the center tile of a multi-tile building needs to be recorded in the blueprint
  - Tile occupancy and validity checking so, for example, buildings that cannot be placed on a target tile will be skipped instead of messing up the blueprint. Blueprints that are only partially applied for any reason (e.g. you need to dig out some more tiles) can be safely reapplied to build the remaining buildings.
  - Relaxed rules for farm plot and road placement: you can still place the building even if an invalid tile (e.g. stone tiles for farm plots) splits the designated area into two disconnected parts
  - Intelligent boundary detection for adjacent buildings of the same type (e.g. a 6x6 block of `wj` cells will be correctly split into 4 jeweler's workshops)
- Place and zone modes
  - Define stockpiles and zones of any shape, not just rectangles
  - Configurable numbers of bins, barrels and wheelbarrows assigned to created stockpiles
  - Automatic splitting of stockpiles and zones that exceed maximum dimension limits
  - Fully configurable zone settings, such as pit/pond and hospital supply counts
- Query mode
  - Send arbitrary keystroke sequences to the UI – *anything* you can do through the UI is supported
  - Supports aliases to simplify frequent keystroke combos
  - Includes a library of pre-made and tested aliases to simplify most common tasks, such as configuring stockpiles for important item types or creating hauling routes for quantum stockpiles.
  - Supports expanding aliases in other aliases for easy management of common subsequences
  - Supports repeating key sequences a specified number of times

- Skips sending keys when the cursor is over a tile that does not have a stockpile or building, so missing buildings won't desynchronize your blueprint
- Instant halting of query blueprint application when keystroke errors are detected, such as when a mistake in a key sequence leaves us stuck in a submenu, to make query blueprints easier to debug

7.4.2 Creating blueprints

We recommend using a spreadsheet editor such as Excel, Google Sheets, or LibreOffice to edit blueprint files, but any text editor will do.

The format of Quickfort-compatible blueprint files is straightforward. The first line (or upper-left cell) of the spreadsheet should look like this:

```
#dig
```

The keyword `dig` tells Quickfort we are going to be using the Designations menu in DF. The following "mode" keywords are understood:

Blueprint mode	Description
dig	Designations menu (`d`)
build	Build menu (`b`)
place	Place stockpiles menu (`p`)
zone	Activity zones menu (`i`)
query	Set building tasks/prefs menu (`q`)

If no modeline appears in the first cell, Quickfort assumes that it's looking at a `#dig` blueprint.

There are also other modes that don't directly correspond to Dwarf Fortress menus, but we'll talk about those *later*.

If you like, you may enter a comment after the mode keyword. This comment will appear in the output of `quickfort list` when run from the `DFHack#` prompt or in the dialog window when running *gui/quickfort*. You can use this space for explanations, attribution, etc.:

```
#dig grand dining room
```

Below this line, begin entering keys in each spreadsheet cell that represent what you want designated in the corresponding game map tile. For example, we could dig out a 4x4 room like so (spaces are used as column separators here for readability, but a real .csv file would have commas):

```
#dig
d d d d #
d d d d #
d d d d #
d d d d #
# # # # #
```

Note the `#` symbols at the right end of each row and below the last row. These are completely optional, but can be helpful to make the row and column positions clear.

Once the dwarves have that dug out, let's build a walled-in bedroom within our dug-out area:

```
#build
Cw Cw Cw Cw #
Cw b  h  Cw #
Cw       Cw #
```

(continues on next page)

(continued from previous page)

```
Cw Cw    Cw #
#  #  #  #  #
```

Note my generosity – in addition to the bed (`b`) I've built a chest (`c`) here for the dwarf as well. You must use the full series of keys needed to build something in each cell, e.g. `Cw` indicates we should enter DF's constructions submenu (`C`) and select walls (`w`).

I'd also like to place a booze stockpile in the 2 unoccupied tiles in the room:

```
#place Place a food stockpile
` ` ` ` #
` ~ ~ ` #
` f f ` #
` `   ` #
# # # # #
```

This illustration may be a little hard to understand. The two `f` characters are in row 3, columns 2 and 3. All the other cells are empty. QF considers both ` (backtick – the character under the tilde) and ~ (tilde) characters within cells to be empty cells; this can help with multilayer or fortress-wide blueprint layouts as "chalk lines".

QF is smart enough to recognize this as a 2x1 food stockpile, and creates it as such rather than as two 1x1 food stockpiles. Quickfort treats any connected region of identical designations as a single entity. The tiles can be connected orthogonally or diagonally, just as long as they are touching.

Lastly, let's turn the bed into a bedroom and set the food stockpile to hold only booze.

```
#query
` ` ` ` #
` r&  ` #
` booze #
` ` ` ` #
# # # # #
```

In row 2, column 2 we have `r&`. This sends the `r` key to DF when the cursor is over the bed, causing us to "make room" and `Enter`, represented by special `&` alias, to indicate that we're done setting the size (the default room size is fine here).

In column 2, row 3 we have `booze`. This is one of many alias keywords defined in the included aliases library. This particular alias sets a food stockpile to accept only booze. It sends the keys needed to navigate DF's stockpile settings menu, and then it sends an Escape character to exit back to the map. It is important to exit out of any menus that you enter while in query mode so that the cursor can move to the next tile when it is done with the current tile.

If there weren't an alias named `booze` then the literal characters `booze` would have been sent, so be sure to spell those aliases correctly!

You can save a lot of time and effort by using aliases instead of adding all key seqences directly to your blueprints. For more details, check out the *Quickfort Keystroke Alias Guide*. You can also see examples of aliases being used in the query blueprints in the DFHack blueprint library. You can create your own aliases by adding them to dfhack-config/quickfort/aliases.txt in your DFHack folder or you can package them *together with your blueprint files*.

Area expansion syntax

In Quickfort, the following blueprints are equivalent:

```
#dig a 3x3 area
d d d #
```

(continues on next page)

(continued from previous page)

```
d d d #
d d d #
# # # #

#dig the same area with d(3x3) specified in row 1, col 1
d(3x3)#
` ` ` #
` ` ` #
# # # #
```

The second example uses Quickfort's "area expansion syntax", which takes the form:

```
keys(WxH)
```

Note that area expansion syntax can only specify rectangular areas. If you want to create extent-based structures (e.g. farm plots or stockpiles) in different shapes, use the first format above. For example:

```
#place A single L shaped food stockpile
f f ` ` #
f f ` ` #
f f f f #
f f f f #
# # # # #
```

Area expansion syntax also sets boundaries, which can be useful if you want adjacent, but separate, stockpiles of the same type:

```
#place Two touching but separate food stockpiles
f(2x2)  #
~ ~ ` ` #
f(4x2)  #
~ ~ ~ ~ #
# # # # #
```

As mentioned previously, ~ characters are ignored as comment characters and can be used for visualizing the blueprint layout. This blueprint can be equivalently written as:

```
#place Two touching but separate food stockpiles
f(2x2)  #
~ ~ ` ` #
f f f f #
f f f f #
# # # # #
```

since the area expansion syntax of the upper stockpile prevents it from combining with the lower, freeform syntax stockpile.

Area expansion syntax can also be used for buildings which have an adjustable size, like bridges. The following blueprints are equivalent:

```
#build a 4x2 bridge from row 1, col 1
ga(4x2)  `  #
`  `  `  `  #
#  #  #  #  #

#build a 4x2 bridge from row 1, col 1
```

(continues on next page)

(continued from previous page)

```
ga ga ga ga #
ga ga ga ga #
#  #  #  #  #
```

If it is convenient to do so, you can place the cell with the expansion syntax in any corner of the resulting rectangle. Just use negative numbers to indicate which direction the designation should expand in. For example, the previous blueprint could also be written as:

```
#build a 4x2 bridge from row 2, col 4
`  `  `  `  #
ga(4x-2) `  #
#  #  #  #  #
```

Automatic area expansion

Buildings larger than 1x1, like workshops, can be represented in any of three ways. You can designate just their center tile with empty cells around it to leave room for the footprint, like this:

```
#build a mason workshop in row 2, col 2 that will occupy the 3x3 area
` `  ` #
` wm ` #
` `  ` #
# #  # #
```

Or you can fill out the entire footprint like this:

```
#build a mason workshop
wm wm wm #
wm wm wm #
wm wm wm #
#  #  #  #
```

This format may be verbose for regular workshops, but it can be very helpful for laying out structures like screw pump towers and waterwheels, whose "center point" can be non-obvious.

Or you can use area expansion syntax:

```
#build a mason workshop
wm(3x3)  #
`  `  `  #
`  `  `  #
#  #  #  #
```

This style can be convenient for laying out multiple buildings of the same type. If you are building a large-scale block factory, for example, this will create 20 mason workshops all in a row:

```
#build line of 20 mason workshops
wm(60x3)
```

Quickfort will intelligently break large areas of the same designation into appropriately-sized chunks.

Multilevel blueprints

Multilevel blueprints are accommodated by separating Z-levels of the blueprint with #> (go down one z-level) or #< (go up one z-level) at the end of each floor.

```
#dig Stairs leading down to a small room below
j  `  `  #
`  `  `  #
`  `  `  #
#> #  #  #
u  d  d  #
d  d  d  #
d  d  d  #
#  #  #  #
```

The marker must appear in the first column of the row to be recognized, just like a modeline.

You can go up or down multiple levels by adding a number after the < or >. For example:

```
#dig Two double-level quarries
r(10x10)
#>2
r(10x10)
```

Dig priorities

DF designation priorities are supported for `#dig` blueprints. The full syntax is `[letter][number][expansion]`, where if the `letter` is not specified, `d` is assumed, and if `number` is not specified, `4` is assumed (the default priority). So each of these blueprints is equivalent:

```
#dig dig the interior of the room at high priority
d  d  d  d  d  #
d  d1 d1 d1 d  #
d  d1 d1 d1 d  #
d  d1 d1 d1 d  #
d  d  d  d  d  #
#  #  #  #  #  #

#dig dig the interior of the room at high priority
d  d  d  d  d  #
d  d1(3x3)  d  #
d  `  `  `  d  #
d  `  `  `  d  #
d  d  d  d  d  #
#  #  #  #  #  #

#dig dig the interior of the room at high priority
4  4  4  4  4  #
4  1  1  1  4  #
4  1  1  1  4  #
4  1  1  1  4  #
4  4  4  4  4  #
#  #  #  #  #  #
```

Marker mode

Marker mode is useful for when you want to plan out your digging, but you don't want to dig everything just yet. In `#dig` mode, you can add a `m` before any other designation letter to indicate that the tile should be designated in marker mode. For example, to dig out the perimeter of a room, but leave the center of the room marked for digging later:

```
#dig
d  d  d  d d #
d md md md d #
d md md md d #
d md md md d #
d  d  d  d d #
#  #  #  # # #
```

Then you can use "Toggle Standard/Marking" (`dM`) to convert the center tiles to regular designations at your leisure.

To apply an entire dig blueprint in marker mode, regardless of what the blueprint itself says, you can set the global quickfort setting `force_marker_mode` to `true` before you apply the blueprint.

Note that the in-game UI setting "Standard/Marker Only" (`dm`) does not have any effect on quickfort.

Stockpiles and zones

It is very common to have stockpiles that accept multiple categories of items or zones that permit more than one activity. Although it is perfectly valid to declare a single-purpose stockpile or zone and then modify it with a `#query` blueprint, quickfort also supports directly declaring all the types in the `#place` and `#zone` blueprints. For example, to declare a 20x10 stockpile that accepts both corpses and refuse, you could write:

```
#place refuse heap
yr(20x10)
```

And similarly, to declare a zone that is a pasture, a fruit picking area, and a meeting area all at once:

```
#zone main pasture and picnic area
nmg(10x10)
```

The order of the individual letters doesn't matter. If you want to configure the stockpile from scratch in a `#query` blueprint, you can place unconfigured "custom" stockpiles with (`c`). It is more efficient, though, to place stockpiles using the keys that represent the categories of items that you want to store, and then only use a `#query` blueprint if you need fine-grained customization.

Stockpile bins, barrels, and wheelbarrows

Quickfort has global settings for default values for the number of bins, barrels, and wheelbarrows assigned to stockpiles, but these numbers can be set for individual stockpiles as well.

To set the number of bins, barrels, or wheelbarrows, just add a number after the letter that indicates what type of stockpile it is. For example:

```
#place a stone stockpile with 5 wheelbarrows
s5(3x3)

#place a bar, ammo, weapon, and armor stockpile with 20 bins
bzpd20(5x5)
```

If the specified number exceeds the number of available stockpile tiles, the number of available tiles is used. For wheelbarrows, that limit is reduced by 1 to ensure there is at least one non-wheelbarrow tile available in the stockpile. Otherwise no stone would ever be brought to the stockpile since all tiles would be occupied by wheelbarrows!

Quickfort figures out which container type is being set by looking at the letter that comes just before the number. For example `zf10` means 10 barrels in a stockpile that accepts both ammo and food, whereas `z10f` means 10 bins. If the stockpile category doesn't usually use any container type, like refuse or corpses, wheelbarrows are assumed:

```
#place a corpse stockpile with 3 wheelbarrows
y3(3x3)
```

Note that if you are not using expansion syntax, each tile of the stockpile must have the same text. Otherwise the stockpile boundaries will not be detected properly:

```
#place a non-rectangular animal stockpile with 5 wheelbarrows
a5,a5,a5,a5
a5,  ,  ,a5
a5,  ,  ,a5
a5,a5,a5,a5
```

Running `quickfort orders` on a `#place` blueprint with explicitly set container/wheelbarrow counts will enqueue manager orders for the specified number of containers or wheelbarrows, even if that number exceeds the in-game size of the stockpile. For example, `quickfort orders` on the following blueprint will enqueue 10 rock pots, even though the stockpile only has 9 tiles:

```
#place
f10(3x3)
```

Zone detailed configuration

Detailed configuration for zones, such as the pit/pond toggle, can also be set by mimicking the hotkeys used to set them. Note that gather flags default to true, so specifying them in a blueprint will turn the toggles off. If you need to set configuration from multiple zone subscreens, separate the key sections with `^`. Note the special syntax for setting hospital supply levels, which have no in-game hotkeys:

```
#zone a combination hospital and shrub (but not fruit) gathering zone
gGtf^hH{hospital buckets=5 splints=20}(10x10)
```

The valid hospital settings (and their maximum values) are:

```
thread   (1500000)
cloth    (1000000)
splints  (100)
crutches (100)
plaster  (15000)
buckets  (100)
soap     (15000)
```

To toggle the `active` flag for zones, add an `a` character to the string. For example, to create a *disabled* pond zone (that you later intend to carefully fill with 3-depth water for a dwarven bathtub):

```
#zone disabled pond zone
apPf(1x3)
```

Minecart tracks

There are two ways to produce minecart tracks, and they are handled very differently by the game. You can carve them into hard natural floors or you can construct them out of building materials. Constructed tracks are conceptually simpler, so we'll start with them.

Constructed tracks

Quickfort supports the designation of track stops and rollers in `#build` blueprints. You can build a track stop with `CS` and some number of `d` and `a` characters for selecting dump direction and friction. You can build a roller with `Mr` and some number of `s` and `q` characters for direction and speed. However, this can get confusing very quickly and is very difficult to read in a blueprint. Moreover, constructed track segments don't even have keys associated with them at all!

To solve this problem, Quickfort provides the following keywords for use in build blueprints:

```
-- Track segments --
trackN
trackS
trackE
trackW
trackNS
trackNE
trackNW
trackSE
trackSW
trackEW
trackNSE
trackNSW
trackNEW
trackSEW
trackNSEW

-- Track/ramp segments --
trackrampN
trackrampS
trackrampE
trackrampW
trackrampNS
trackrampNE
trackrampNW
trackrampSE
trackrampSW
trackrampEW
trackrampNSE
trackrampNSW
trackrampNEW
trackrampSEW
trackrampNSEW

-- Horizontal and vertical roller segments --
rollerH
rollerV
rollerNS
rollerSN
rollerEW
rollerWE

Note: append up to four 'q' characters to roller keywords to set roller
speed. E.g. a roller that propels from East to West at the slowest speed can
be specified with 'rollerEWqqqq'.

-- Track stops that (optionally) dump to the N/S/E/W --
```

(continues on next page)

(continued from previous page)

```
trackstop
trackstopN
trackstopS
trackstopE
trackstopW

Note: append up to four 'a' characters to trackstop keywords to set friction
amount. E.g. a stop that applies the smallest amount of friction can be
specified with 'trackstopaaaa'.
```

As an example, you can create an E-W track with stops at each end that dump to their outside directions with the following blueprint:

```
#build Example track
trackstopW trackEW trackEW trackEW trackstopE
```

Note that the **only** way to build track and track/ramp segments is with the keywords. The UI method of using + and − keys to select the track type from a list does not work since DFHack Quickfort doesn't actually send keys to the UI to build buildings. The text in your spreadsheet cells is mapped directly onto DFHack API calls. Only `#query` blueprints send actual keycodes to the UI.

Carved tracks

In the game, you carve a minecart track by specifying a beginning and ending tile and the game "adds" the designation to the tiles in between. You cannot designate single tiles because DF needs a multi-tile track to figure out which direction the track should go on each tile. For example to carve two track segments that cross each other, you might use the cursor to designate a line of three vertical tiles like this:

```
` start here ` #
` `          ` #
` end here   ` #
# #          # #
```

Then to carve the cross, you'd do a horizonal segment:

```
`           ` `         #
start here ` end here #
`           ` `         #
#           # #         #
```

This will result in a carved track that would be equivalent to a constructed track of the form:

```
#build
`       trackS     `       #
trackE trackNSEW trackW #
`       trackN     `       #
#       #          #       #
```

Quickfort supports both styles of specification for carving tracks with `#dig` blueprints. You can use the "additive" style to carve tracks in segments or you can use the aliases to specify the track tile by tile. To designate track segments, use area expansion syntax with a height or width of 1:

```
#dig
`       T(1x3) ` #
```

(continues on next page)

(continued from previous page)

```
T(3x1) `       ` #
`      `       ` #
#      #       # #
```

"But wait!", I can hear you say, "How do you designate a track corner that opens to the South and East? You can't put both T(1xH) and T(Wx1) in the same cell!" This is true, but you can specify both width and height greater than 1, and for tracks, QF interprets it as an upper-left corner extending to the right W tiles and down H tiles. For example, to carve a track in a closed ring, you'd write:

```
#dig
T(3x3) ` T(1x3) #
`      ` `      #
T(3x1) ` `      #
#      # #      #
```

You can also use negative numbers in the expansion syntax to indicate corners that are not upper-left corners. This blueprint will also carve a closed ring:

```
#dig
T(3x3) ` `         #
`      ` `         #
`      ` T(-3x-3)  #
#      # #         #
```

Or you could use the aliases to specify tile by tile:

```
#dig
trackSE trackEW trackSW #
trackNS `       trackNS #
trackNE trackEW trackNW #
#       #       #       #
```

The aliases can also be used to designate a solid block of track. This is epecially useful for obliterating low-quality engravings so you can re-smooth and re-engrave with higher quality. For example, you could use the following sequence of blueprints to ensure a 10x10 floor area contains only masterwork engravings:

```
#dig smooth floor
s(10x10)
#dig engrave floor
e(10x10)
#dig erase low-quality engravings
trackNSEW(10x10)
```

The tracks only remove low-quality engravings since quickfort won't designate masterwork engravings for destruction (unless forced to by a commandline parameter). You would run (and let your dwarves complete the jobs for) the sequence of blueprints until no tiles are designated by the "erase" blueprint.

Modeline markers

The modeline has some additional optional components that we haven't talked about yet. You can:

- give a blueprint a label by adding a `label()` marker
- set a cursor offset and/or cursor placement hint by adding a `start()` marker
- hide a blueprint from being listed with a `hidden()` marker

- register a message to be displayed after the blueprint is successfully applied with a `message()` marker

The full modeline syntax, when all optional elements are specified, is:

```
#mode label(mylabel) start(X;Y;STARTCOMMENT) hidden() message(mymessage) comment
```

Note that all elements are optional except for the initial `#mode` (though, as mentioned in the first section, if a modeline doesn't appear at all in the first cell of a spreadsheet, the blueprint is interpreted as a `#dig` blueprint with no optional markers). Here are a few examples of modelines with optional elements before we discuss them in more detail:

```
#dig start(3; 3; Center tile of a 5-tile square) Regular blueprint comment
#build label(noblebedroom) start(10;15)
#query label(configstockpiles) No explicit 'start()' means cursor is at upper left␣
↪corner
#meta label(digwholefort) start(center of stairs on surface)
#dig label(digdining) hidden() called by the digwholefort meta blueprint
#zone label(pastures) message(remember to assign animals to the new pastures)
```

Blueprint labels

Labels are displayed in the `quickfort list` output and are used for addressing specific blueprints when there are multiple blueprints in a single file or spreadsheet sheet (see *Packaging a set of blueprints* below). If a blueprint has no label, the label becomes the ordinal of the blueprint's position in the file or sheet. For example, the label of the first blueprint will be "1" if it is not otherwise set, the label of the second blueprint will be "2" if it is not otherwise set, etc. Labels that are explicitly defined must start with a letter to ensure the auto-generated labels don't conflict with user-defined labels.

Start positions

Start positions specify a cursor offset for a particular blueprint, simplifying the task of blueprint alignment. This is very helpful for blueprints that are based on a central staircase, but it comes in handy whenever a blueprint has an obvious "center". For example:

```
#build start(2;2;center of workshop) label(masonw) a mason workshop
wm wm wm #
wm wm wm #
wm wm wm #
#  #  #  #
```

will build the workshop *centered* on the cursor, not down and to the right of the cursor.

The two numbers specify the column and row (or 1-based X and Y offset) where the cursor is expected to be when you apply the blueprint. Position `1;1` is the top left cell. The optional comment will show up in the `quickfort list` output and should contain information about where to position the cursor. If the start position is `1;1`, you can omit the numbers and just add a comment describing where to put the cursor. This is also useful for meta blueprints that don't actually care where the cursor is, but that refer to other blueprints that have fully-specified `start()` markers. For example, a meta blueprint that refers to the `masonw` blueprint above could look like this:

```
#meta start(center of workshop) a mason workshop
/masonw
```

You can use semicolons, commas, or spaces to separate the elements of the `start()` marker, whatever is most convenient.

Hiding blueprints

A blueprint with a `hidden()` marker won't appear in `quickfort list` output unless the `--hidden` flag is specified. The primary reason for hiding a blueprint (rather than, say, deleting it or moving it out of the `blueprints/` folder) is if a blueprint is intended to be run as part of a larger sequence managed by a *meta blueprint*.

Messages

A blueprint with a `message()` marker will display a message after the blueprint is applied with `quickfort run`. This is useful for reminding players to take manual steps that cannot be automated, like assigning minecarts to a route, or listing the next step in a series of blueprints. For long or multi-part messages, you can embed newlines:

```
"#meta label(surface1) message(This would be a good time to start digging the␣
↪industry level.
Once the area is clear, continue with /surface2.) clear the embark site and set up␣
↪pastures"
```

The quotes surrounding the cell text are only necessary if you are writing a .csv file by hand. Spreadsheet applications will surround multi-line text with quotes automatically when they save/export the file.

Meta blueprints

Meta blueprints are blueprints that control how other blueprints are applied. For example, meta blueprints can bundle a group of other blueprints so that they can be run with a single command. They can also encode logic, like rotating the blueprint or duplicating it across a specified number of z-levels.

A common scenario where meta blueprints are useful is when you have several phases to link together. For example you might:

1. Apply a dig blueprint to designate dig areas
2. Wait for miners to dig
3. **Apply a build buildprint** to designate buildings
4. **Apply a place buildprint** to designate stockpiles
5. **Apply a query blueprint** to configure stockpiles
6. Wait for buildings to get built
7. Apply a different query blueprint to configure rooms

Those three "apply"s in the middle might as well get done in one command instead of three. A `#meta` blueprint can help with that. A meta blueprint refers to other blueprints in the same file by their label (see the *Modeline markers* section above) in the same format used by the *quickfort* command: `<sheet name>/<label>`, or just `/<label>` for blueprints in .csv files or blueprints in the same spreadsheet sheet as the `#meta` blueprint that references them.

A few examples might make this clearer. Say you have a .csv file with blueprints that prepare bedrooms for your dwarves:

```
#dig label(bed1) dig out the rooms
...
#build label(bed2) build the furniture
...
#place label(bed3) add food stockpiles
...
```

(continues on next page)

(continued from previous page)

```
#query label(bed4) configure stockpiles
...
#query label(bed5) set the built beds as rooms
...
```

Note how I've given them all labels so we can address them safely. If I hadn't given them labels, they would receive default labels of "1", "2", "3", etc, but those labels would change if I ever add more blueprints at the top. This is not a problem if we're just running the blueprints individually from the `quickfort list` command, but meta blueprints need a label name that isn't going to change over time.

So let's add a meta blueprint to this file that will combine the middle three blueprints into one:

```
"#meta label(bed234) combines build, place, and stockpile config blueprints"
/bed2
/bed3
/bed4
```

Now your sequence is shortened to:

1. Run /bed1 to designate dig areas
2. Wait for miners to dig
3. **Run /bed234 meta buildprint** to build buildings and designate/configure stockpiles
4. Wait for buildings to get built
5. Run /bed5 to configure the rooms as bedrooms

You can use meta blueprints to lay out your fortress at a larger scale as well. The #< and #> notation is valid in meta blueprints, so you can, for example, store the dig blueprints for all the levels of your fortress in different sheets in a spreadsheet, and then use a meta blueprint to designate your entire fortress for digging at once. For example, say you have a .xlsx spreadsheet with the following layout:

Sheet name	Contents
dig_farming	one #dig blueprint, no label
dig_industry	one #dig blueprint, no label
dig_dining	four #dig blueprints, with labels "main", "basement", "waterway", and "cistern"
dig_guildhall	one #dig blueprint, no label
dig_suites	one #dig blueprint, no label
dig_bedrooms	one #dig blueprint, no label

We can add a sheet named "dig_all" with the following contents (we're expecting a big fort, so we're digging 5 levels of bedrooms):

```
#meta dig the whole fortress
dig_farming/1
#>
dig_industry/1
#>
dig_dining/main
#>
dig_dining/basement
#>
dig_dining/waterway
#>
dig_dining/cistern
```

(continues on next page)

(continued from previous page)

```
#>
dig_guildhall/1
#>
dig_suites/1
#>
dig_bedrooms/1 repeat(down 5)
```

Note that for blueprints without an explicit label, we still need to address them by their auto-generated numeric label.

The command to run the meta blueprint above would be:

```
quickfort run myfort.xlsx -n dig_all
```

It's worth repeating that `#meta` blueprints can only refer to blueprints that are defined in the same file. This means that all blueprints that a meta blueprint needs to run must be in sheets within the same .xlsx spreadsheet or concatenated into the same .csv file.

You can then hide the blueprints that you now manage with the meta blueprint from `quickfort list` by adding a `hidden()` marker to their modelines. That way the output of `quickfort list` won't be cluttered by blueprints that you don't need to run directly. If you ever *do* need to access the meta-managed blueprints individually, you can still see them with `quickfort list --hidden`.

Meta markers

In meta blueprints, you can tag referenced blueprints with markers to modify how they are applied. These markers are similar to *Modeline markers*, but are only usable in meta blueprints. Here's a quick list of examples, with more details below:

Example	Description
repeat(down 10)	Repeats a blueprint down z-levels 10 times
shift(0 10)	Adds 10 to the y coordinate of each blueprint tile
transform(cw flipv)	Rotates a blueprint clockwise and then flips it vertically

Repeating blueprints

Syntax: repeat(<direction>[,]<number>)

The direction can be `up` or `down`, and the repetition works even for blueprints that are themselves multi-level. For example:

```
#meta label(2beds) dig 2 levels of bedrooms
dig_bedrooms/1 repeat(down 2)

#meta label(6beds) dig 6 levels of bedrooms
/2beds repeat(down 3)
```

You can use < and > for short, instead of `up` and `down`. The comma or space between the direction and the number is optional as well. The following lines are all equivalent:

```
/2beds repeat(down 3)
/2beds repeat(down, 3)
/2beds repeat(>3)
```

Shifting blueprints

Syntax: shift(<x shift>[[,] <y shift>])

The values can be positive or negative. Negative values for x shift to the left, positive to the right. Negative values for y shift up, positive down. Note the semantics for the y axis are opposite compared to regular graphs on paper. This is because the y coordinates in the DF game map start a 0 at the top and increase as they go down.

Transforming blueprints

Syntax: transform(<transformation>[[,] <transformation>...])

Applies a geometric transformation to the blueprint. The supported transformations are:

rotcw or cw Rotates the blueprint 90 degrees clockwise.

rotccw or ccw Rotates the blueprint 90 degrees counterclockwise.

fliph Flips the blueprint horizontally (left edge becomes right edge).

flipv Flips the blueprint vertically (top edge becomes bottom edge).

If you specify more than one transformation, they will be applied in the order they appear in.

If you use both `shift()` and `transform()` markers on the same blueprint, shifting is applied after all transformations are complete. If you want shifting to be applied before the transformations, or in between transformations, you can use nested meta blueprints. For example, the following blueprint will shift the `/hallway` blueprint to the right by 20 units and then rotate it clockwise:

```
#meta label(shift_right) hidden()
/hallway shift(20)
#meta label(rotate_after_shift)
/shift_right transform(cw)
```

Transforming build blueprints will also change the properties of buildings that care about direction. For example, a bridge that opens to the North, rotated clockwise, will open to the East when applied to the map.

Direction keys that move the cursor on the map will also be transformed. For example, the keys `g{Up 4}&` that would cause a stockpile to give to a workshop 4 tiles to the North become `g{Right 4}&` when played back on a clockwise-rotated `#query` blueprint. Direction keys that don't move the map cursor, for example when on the stockpile configuration screen, are not changed by blueprint rotation.

Other blueprint modes

In addition to the powerful `#meta` mode described above, there are a few additional blueprint modes that become useful when you are sharing your blueprints with others or managing complex blueprint sets. Instead of mapping tile positions to map modifications like the basic modes do, these "blueprints" have specialized, higher-level uses:

Blueprint mode	Description
config	Play back key sequences that are not related to map tiles
notes	Display long messages, such as help text or blueprint walkthroughs
aliases	Define aliases that can be used by other `#query` blueprints in the same file
ignore	Hide a section of your spreadsheet from quickfort, useful for scratch space or personal notes

Config blueprints

A `#config` blueprint is used to send unfiltered keystrokes directly to the DF UI without interacting with specific map tiles. They have access to the same keystroke aliases as `#query` blueprints, but `#config` blueprints differ from `#query` blueprints in a few critical ways:

- Whereas the "home" mode for `#query` blueprints is the "query" mode (`q`), `#config` blueprints start on the default map screen – the view you have when you're looking at the map with no sidebar visible. The keystroke or alias sequence in each spreadsheet cell in a `#config` blueprint must begin and end on the default map screen.
- The cursor position is not set for `#config` blueprints. This means that it doesn't matter what spreadsheet cell you put your text in. The blueprint cell location does not correspond to a map tile.

A `#config` blueprint is best used for accessing game menus that are not associated with map tiles, such as the hotkey menu (`H`), the military menu (`m`), or the standing orders menu (`o`). In other words, use a `#config` blueprint when you want to configure the game itself, not the tiles on the map. A `#config` blueprint is better for these menus than a `#query` blueprint because the cursor can jump around in unpredictable ways when accessing these non-cursor modes and then re-entering query mode. This will cause quickfort to detect a `#query` blueprint error and stop executing. Also, `#query` blueprints will skip playing back a key sequence entirely if it doesn't detect a building or zone on the target tile. A `#config` blueprint doesn't need a building or zone to exist in order to run.

Note that you *can* enter any mode you want during a `#config` blueprint keystroke sequence (as long as you get back to the default map screen by the end of the sequence), even modes that provide a cursor on the screen. It's just that the position of that cursor is not guaranteed to be on any specific tile. If you need access to a cursor, you probably should be using a `#query` blueprint instead.

Notes blueprints

Sometimes you just want to record some information about your blueprints, such as when to apply them, what preparations you need to make, or what the blueprints contain. The *message()* modeline marker is useful for small, single-line messages, but a `#notes` blueprint is more convenient for long messages or messages that span many lines. The lines in a `#notes` blueprint are output as if they were contained within one large multi-line `message()` marker. For example, the following (empty) `#meta` blueprint:

```
"#meta label(help) message(This is the help text for the blueprint set
contained in this file.

First, make sure that you embark in...) blueprint set walkthough"
```

could more naturally be written as a `#notes` blueprint:

```
#notes label(help) blueprint set walkthrough
This is the help text for the blueprint set
contained in this file

First, make sure that you embark in...
```

The `#meta` blueprint is all squashed into a single spreadsheet cell, using embedded newlines. Each line of the `#notes` "blueprint", however, is in a separate cell, allowing for much easier viewing and editing.

Aliases blueprints

There are keystroke aliases that *come with DFHack* that are usable by all blueprints, and you have the ability to define custom aliases in dfhack-config/quickfort/aliases.txt that are visible to all your blueprints as well. An `#aliases` blueprint can define custom aliases that are only visible to the current `.csv` or `.xlsx` file. Packaging aliases in the same file that uses them is convenient for specialized aliases that are only useful to a particular blueprint. Also, if you want to share your blueprint with others, defining your aliases in an `#aliases` blueprint will help your blueprint to work "out of the box", and you won't need others to add your custom aliases to their `dfhack-config/quickfort/aliases.txt` files.

Although we're calling them "blueprints", `#aliases` blueprints are not actual blueprints, and they don't show up when you run `quickfort list`. The aliases are just automatically read in when you run any `#query` blueprint that is defined in the same file.

Aliases can be in either of two formats, and you can mix formats freely within the same `#aliases` section. The first format is the same as what is used in the `aliases.txt` files:

```
#aliases
aliasname: aliasdefinition
```

Aliases in this format must appear in the first column of a row.

The second format has the alias name in the first column and the alias definition in the second column, with no `:` separator:

```
#aliases
aliasname,aliasdefinition
```

There can be multiple #aliases sections defined in a .csv file or .xlsx spreadsheet. The aliases are simply combined into one list. If an alias is defined multiple times, the last definition wins.

See the *Quickfort Keystroke Alias Guide* for help with the alias definition syntax.

Ignore blueprints

If you don't want some data to be visible to quickfort at all, use an `#ignore` blueprint. All lines until the next modeline in the file or sheet will be completely ignored. This can be useful for personal notes, scratch space, or temporarily "commented out" blueprints.

7.4.3 Packaging a set of blueprints

A complete specification for a section of your fortress may contain 5 or more separate blueprints, one for each "phase" of construction (dig, build, place stockpiles, designate zones, and query adjustments).

To manage all the separate blueprints, it is often convenient to keep related blueprints in a single file. For .xlsx spreadsheets, you can keep each blueprint in a separate sheet. Online spreadsheet applications like Google Sheets make it easy to work with multiple related blueprints, and, as a bonus, they retain any formatting you've set, like column sizes and coloring.

For both .csv files and .xlsx spreadsheets you can also add as many blueprints as you want in a single file or sheet. Just add a modeline in the first column to indicate the start of a new blueprint. Instead of multiple .csv files, you can concatenate them into one single file. This is especially useful when you are sharing your blueprints with others. A single file is much easier to manage than a directory of files.

For example, you can write multiple blueprints in one file like this:

```
#dig label(bed1)
d d d d #
d d d d #
d d d d #
d d d d #
# # # # #
#build label(bed2)
b   f h #
        #
        #
```

(continues on next page)

(continued from previous page)

```
n        #
# # # # #
#place label(bed3)
         #
f(2x2)   #
         #
         #
# # # # #
#query label(bed4)
         #
booze    #
         #
         #
# # # # #
#query label(bed5)
r{+ 3}& #
         #
         #
         #
# # # # #
```

Of course, you could still choose to keep your blueprints in separate files and just give related blueprints similar names:

```
bedroom.1.dig.csv
bedroom.2.build.csv
bedroom.3.place.csv
bedroom.4.query.csv
bedroom.5.query2.csv
```

The naming and organization is completely up to you.

7.4.4 Buildingplan integration

Buildingplan is a DFHack plugin that keeps building construction jobs in a suspended state until the materials required for the job are available. This prevents a building designation from being canceled when a dwarf picks up the job but can't find the materials.

As long as the *buildingplan* plugin is enabled, quickfort will use it to manage construction. The buildingplan plugin has an *"enabled" setting* for each building type, but those settings only apply to buildings created through the buildingplan user interface. Quickfort will still use buildingplan to plan buildings even if the buildingplan UI says that building type is not "enabled".

In addition, buildingplan has a "quickfort_mode" setting for compatibility with legacy Python Quickfort. This setting has no effect on DFHack Quickfort, which will use buildingplan to manage everything designated in a `#build` blueprint regardless of the buildingplan UI settings.

However, quickfort *does* use *buildingplan's filters* for each building type. For example, you can use the buildingplan UI to set the type of stone you want your walls made out of. Or you can specify that all buildingplan-managed chairs and tables must be of Masterful quality. The current filter settings are saved with planned buildings when the `#build` blueprint is run. This means you can set the filters the way you want for one blueprint, run the blueprint, and then freely change them again for the next blueprint, even if the first set of buildings haven't been built yet.

Note that buildings are still constructed immediately if you already have the materials. However, with buildingplan you now have the freedom to apply `#build` blueprints before you manufacture the resources. The construction jobs will be fulfilled whenever the materials become available.

Since it can be difficult to figure out exactly what source materials you need for a `#build` blueprint, quickfort supplies the `orders` command. It enqueues manager orders for everything that the buildings in a `#build` blueprint require. See the next section for more details on this.

Alternately, if you know you only need a few types of items, the *workflow* plugin can be configured to build those items continuously for as long as they are needed.

If you do not want to enable the buildingplan plugin, run `quickfort orders` and make sure all manager orders are fulfilled before applying a `#build` blueprint. Otherwise you will get job cancellation spam when the buildings can't be built with available materials.

7.4.5 Generating manager orders

Quickfort can generate manager orders to make sure you have the proper items in stock for a `#build` blueprint.

Many items can be manufactured from different source materials. Orders will always choose rock when it can, then wood, then cloth, then iron. You can always remove orders that don't make sense for your fort and manually enqueue a similar order more to your liking. For example, if you want silk ropes instead of cloth ropes, make a new manager order for an appropriate quantity of silk ropes, and then remove the generated cloth rope order.

Anything that requires generic building materials (workshops, constructions, etc.) will result in an order for a rock block. One "Make rock blocks" job produces four blocks per boulder, so the number of jobs ordered will be the number of blocks you need divided by four (rounded up). You might end up with a few extra blocks, but not too many.

If you want your constructions to be in a consistent color, be sure to choose a rock type for all of your 'Make rock blocks' orders by selecting the order and hitting `d`. You might want to set the rock type for other non-block orders to something different if you fear running out of the type of rock that you want to use for blocks. You should also set the *buildingplan* material filter for construction building types to that type of rock as well so other blocks you might have lying around aren't used.

Extra Manager Orders

In `#build` blueprints, there are a few building types that will generate extra manager orders for related materials:

- Track stops will generate an order for a minecart
- Traction benches will generate orders for a table, mechanism, and rope
- Levers will generate an order for an extra two mechanisms for connecting the lever to a target
- Cage traps will generate an order for a cage

Stockpiles in `#place` blueprints that *specify wheelbarrow or container counts* will generate orders for the appropriate number of bins, pots, or wheelbarrows.

7.4.6 Tips and tricks

- During blueprint application, especially query blueprints, don't click the mouse on the DF window or type any keys. They can change the state of the game while the blueprint is being applied, resulting in strange errors.
- After digging out an area, you may wish to smooth and/or engrave the area before starting the build phase, as dwarves may be unable to access walls or floors that are behind/under built objects.
- If you are designating more than one level for digging at a time, you can make your miners more efficient by using marker mode on all levels but one. This prevents your miners from digging out a few tiles on one level, then running down/up the stairs to do a few tiles on an adjacent level. With only one level "live" and all other levels in marker mode, your miners can concentrate on one level at a time. You just have to remember to

"unmark" a new level when your miners are done with their current one. Alternately, if you have a chokepoint between levels (e.g. a central staircase), you can set the chokepoint to be dug at a lower priority than all the other tiles on the level. This will ensure your miners complete digging out a level before continuing on to the next.

- As of DF 0.34.x, it is no longer possible to build doors (`d`) at the same time that you build adjacent walls (`Cw`). Doors must now be built *after* adjacent walls are constructed. This does not affect the more common case where walls exist as a side-effect of having dug-out a room in a `#dig` blueprint, but if you are building your own walls, be aware that walls must be built before you run the blueprint to designate attached doors.
- Quickfort is a very powerful tool. See the *case study* below for more ideas on how to build awesome blueprints!

7.4.7 Caveats and limitations

- If you use the `jugs` alias in your `#query`-mode blueprints, be aware that there is no way to differentiate jugs from other types of tools in the game. Therefore, `jugs` stockpiles will also take nest boxes, scroll rollers, and other tools. The only workaround is not to have other tools lying around in your fort.
- Likewise for the `bags` alias. The game does not differentiate between empty and full bags, so you'll get bags of gypsum power in your "bags" stockpile unless you are careful to assign all your gypsum to your hospital.
- Weapon traps and upright spear/spike traps can currently only be built with a single weapon.
- Pressure plates can be built, but they cannot be usefully configured yet.
- Building instruments is not yet supported.
- DFHack Quickfort is a large project, and there are bound to be bugs! Please report them at the DFHack issue tracker so they can be addressed.

7.4.8 Dreamfort case study: a practical guide to advanced blueprint design

While syntax definitions and toy examples will certainly get you started with your blueprints, it may not be clear how all the quickfort features fit together or what the best practices are, especially for large and complex blueprint sets. This section walks through the "Dreamfort" blueprints found in the *DFHack blueprint library*, highlighting design choices and showcasing practical techniques that can help you create better blueprints. Note that this is not a guide for how to design the best *fort* (there is plenty about that on the wiki). This is essentially an extended tips and tricks section focused on how to make usable and useful quickfort blueprints that will save you time and energy.

Almost every quickfort feature is used somewhere in Dreamfort, so the blueprints are very useful as reference examples. You can copy the Dreamfort blueprints and use them as starting points for your own, or just refer to them when you create something similar.

In this case study, we'll start by discussing the high level organization of the Dreamfort blueprint set. Then we'll walk through the spreadsheets for each of the fort levels in turn, calling out feature usage examples and explaining the parts that might not be obvious just from looking at them.

If you haven't built Dreamfort before, maybe try an embark in a flat area and take it for a spin! It will help put the following sections in context. There is also a pre-built Dreamfort available for download on dffd if you just want an interactive reference.

Dreamfort organization and packaging

The Dreamfort blueprints are distributed with DFHack as one large .csv file, but editing in that format would be frustrating. Instead, the blueprints are edited online as Google drive spreadsheets. Either the .csv file or the .xlsx files can be read and applied by quickfort, but it made more sense to distribute the blueprints as a .csv so users would only

have to remember one filename. Also, .csv files are text-based, which works more naturally with the DFHack source control system. We use the xlsx2csv utility to do the conversion from .xlsx to .csv format.

Tip

Include a `#notes` section with information about how to use your blueprint.

Each spreadsheet has a "help" sheet with a `#notes` blueprint that displays a walkthrough and other useful details. This is the first sheet in each spreadsheet so it will be selected by default if the user doesn't specify a label name. For example, just running `quickfort run library/dreamfort.csv` will display Dreamfort's introduction text.

Do not neglect writing the help text! Not only will it give others a chance to use your blueprints appropriately, but the help you write will remind *you* what you were thinking when you wrote the blueprint in the first place.

Tip

Include custom alias definitions in the same file as the blueprint.

If any blueprint in the set uses custom aliases that other users won't have in their data/quickfort/aliases-common.txt files, be sure to define them in the blueprint itself in an *Aliases blueprints* section. Then other people can use your blueprint right away without having to manually copy aliases into their personal dfhack-config/quickfort/aliases.txt files.

The surface level: how to manage complexity

For smaller blueprints, packaging and usability are not really that important - just write it, run it, and you're done. However, as your blueprints become larger and more detailed, there are some best practices that can help you deal with the added complexity. Dreamfort's surface level is many steps long since there are trees to be cleared, holes to be dug, flooring to be laid, and bridges to be built, and each step requires the previous step to be completely finished before it can begin. Therefore, a lot of thought went into minimizing the toil associated with applying so many blueprints.

Tip

Use meta blueprints to script blueprint sequences and reduce the number of quickfort commands you have to run.

The single most effective way to make your blueprint sets easier to use is to group them with *meta blueprints*. For the Dreamfort set of blueprints, each logical "step" generally takes more than one blueprint. For example, with `#meta` blueprints, setting up pastures with a `#zone` blueprint, placing starting stockpiles with a `#place` blueprint, building starting workshops with a `#build` blueprint, and configuring the stockpiles with a `#query` blueprint can all be done with a single command. Bundling blueprints with `#meta` blueprints reduced the number of steps in Dreamfort from 61 to 30, and it also made it much clearer to see which blueprints can be applied at once without unpausing the game. Check out dreamfort_surface's "meta" sheet to see how much meta blueprints can simplify your life.

You can define *as many blueprints as you want* on one sheet, but this is especially useful when writing meta blueprints. It's like having a bird's eye view of your entire plan in one sheet.

Tip

Keep the blueprint list uncluttered by using `hidden()` markers.

If a blueprint is bundled into a meta blueprint, it does not need to appear in the `quickfort list` output since you won't be running it directly. Add a *hidden() marker* to those blueprints to keep the list output tidy. You can still access hidden blueprints with `quickfort list --hidden` if you need to – for example to reapply a partially completed `#build` blueprint – but now they won't clutter up the normal blueprint list.

Tip

Name your blueprints with a common prefix so you can find them easily.

This goes for both the file name and the *modeline label()*. Searching and filtering is implemented for both the `quickfort list` command and the quickfort interactive dialog. If you give related blueprints a common prefix, it makes it easy to set the filters to display just the blueprints that you're interested in. If you have a lot of blueprints, this can save you a lot of time. Dreamfort uses the level name as a prefix for the labels, like "surface1", "surface2", "farming1", etc. So if I'm in the middle of applying the surface blueprints, I'd set the filter to `dreamfort surface` to just display the relevant blueprints.

Tip

Add descriptive comments that remind you what the blueprint contains.

If you've been away from Dwarf Fortress for a while, it's easy to forget what your blueprints actually do. Make use of *modeline comments* so your descriptions are visible in the blueprint list. If you use meta blueprints, all your comments can be conveniently edited on one sheet, like in surface's meta sheet.

Tip

Use `message()` markers to remind yourself what to do next.

Messages are displayed after a blueprint is applied. Good things to include in messages are:

- The name of the next blueprint to apply and when to run it
- Whether `quickfort orders` should be run for the current or an upcoming step
- Any actions that you have to perform manually after running the blueprint, like assigning minecarts to hauling routes or pasturing animals in newly-created zones

These things are just too easy to forget. Adding a `message()` can save you from time-wasting mistakes. Note that `message()` markers can still appear on the `hidden()` blueprints, and they'll still get shown when the blueprint is run via a `#meta` blueprint. For an example of this, check out the zones sheet where the pastures are defined.

The farming level: fun with stockpiles

It is usually convenient to store closely associated blueprints in the same spreadsheet. The farming level is very closely tied to the surface because the miasma vents dug on the surface have to perfectly line up with where waste products are placed on the farming level. However, surface is a separate z-level and, more importantly, already has many many blueprints of its own. Farming is therefore split into a separate file.

Tip

Automate stockpile chains when you can, and write `message()` reminders when you can't.

The farming level starts doing interesting things with `#query` blueprints and stockpiles. Note the careful customization of the food stockpiles and the stockpile chains set up with the `give*` aliases. This is so when multiple stockpiles can hold the same item, the largest can keep the smaller ones filled. For example the `give2up` alias funnels seeds from the seeds feeder pile to the container-enabled seed storage pile. If you have multiple stockpiles holding the same type on different z-levels, though, this can be tricky to set up with a blueprint. Here, the jugs and pots stockpiles must be manually linked to the quantum stockpile on the industry level, since we can't know beforehand how many z-levels away that is. Note how we call that out in the `#query` blueprint's `message()`.

Tip

Use aliases to set up hauling routes and quantum stockpiles.

Hauling routes are notoriously fiddly to set up, but they can be automated with blueprints. Check out the Southern area of the `#place` and `#query` blueprints for how the quantum refuse dump is configured with simple aliases from the alias library.

The industry level: when not to use aliases

The industry level is densely packed and has more complicated examples of stockpile configurations and quantum dumps. However, what I'd like to call out first are the key sequences that are *not* in aliases.

Tip

Don't use aliases for ad-hoc cursor movements.

It may be tempting to put all query blueprint key sequences into aliases to make them easier to edit, keep them all in one place, and make them reusable, but some key sequences just aren't very valuable as aliases.

Check out the Eastern (goods) and Northern (stone and gems) quantum stockpiles – cells I19 and R10. They give to the jeweler's workshop to prevent the jeweler from using the gems held in reserve for strange moods. The keys are not aliased since they're dependent on the relative positions of the tiles where they are interpreted, which is easiest to see in the blueprint itself. Also, if you move the workshop, it's easier to fix the stockpile link right there in the blueprint instead of editing a separate alias definition.

There are also good examples in the `#query` blueprint for how to use the `permit` and `forbid` stockpile aliases.

Tip

Put all configuration that must be applied in a particular order in the same spreadsheet cell.

Most of the baseline aliases distributed with DFHack fall into one of three categories:

1. Make a stockpile accept only a particular item type in a category
2. Permit an item type, but do not otherwise change the stockpile configuration
3. Forbid an item type, but do not otherwise change the stockpile configuration

If you have a stockpile that covers multiple tiles, it might seem natural to put one alias per spreadsheet cell. The aliases still all get applied to the stockpile, and with only one alias per cell, you can just type the alias name and avoid having to use the messier-looking `{aliasname}` syntax:

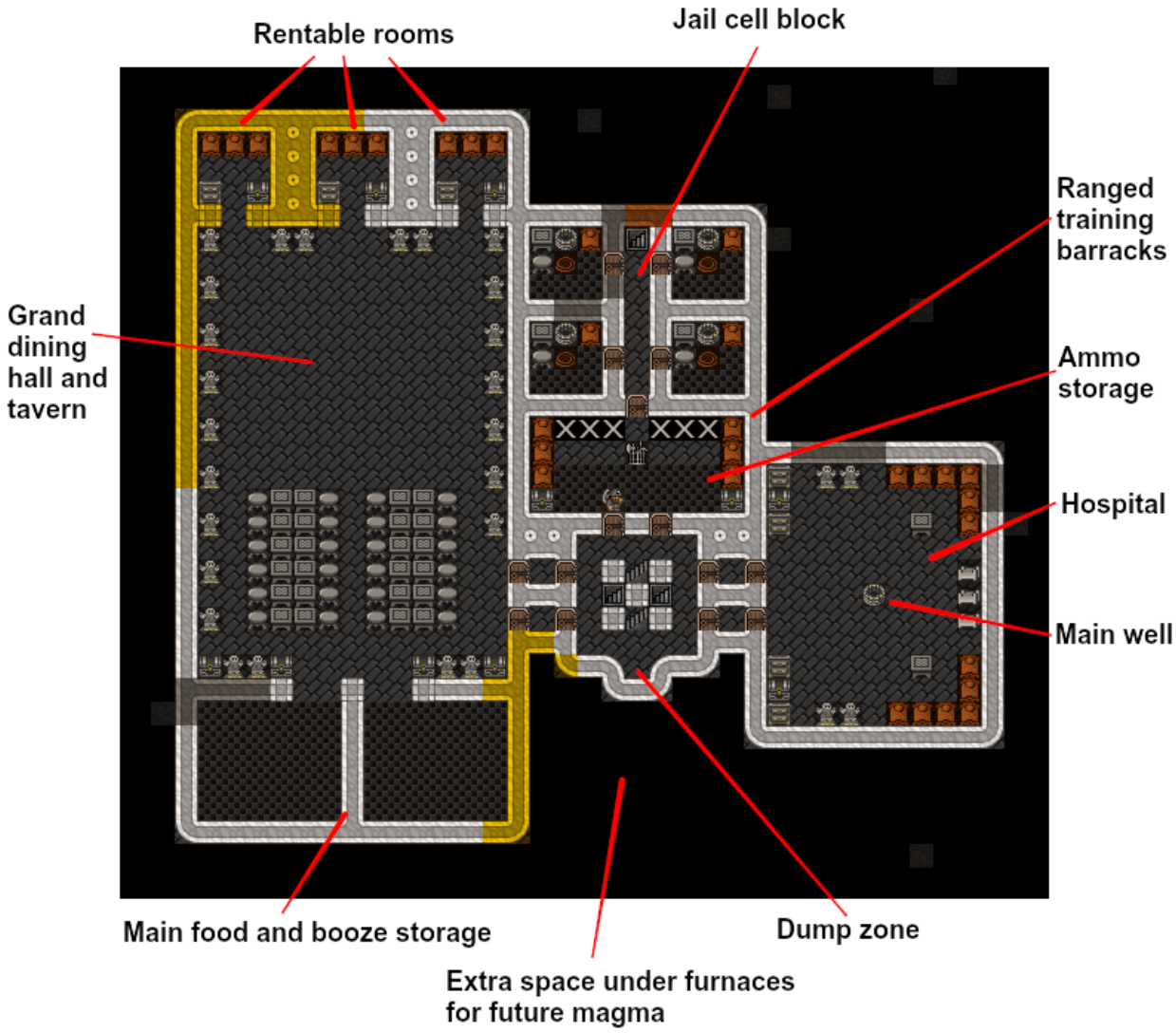
```
#place Declare a food stockpile
f(3x3)
#query Incorrectly configure a food stockpile to accept tallow and dye
tallow
permitdye
```

However, in quickfort there are no guarantees about which cell will be processed first. In the example above, we obviously intend for the food stockpile to have tallow exclusively permitted, then to add dye. It could happen that the two aliases are applied in the opposite order, though, and we'd end up with dye being permitted, then everything (including dye) being forbidden, and, finally, tallow being enabled. To make sure you always get what you want, write order-sensitive aliases on the same line:

```
#place Declare a food stockpile
f(3x3)
#query Properly configure a food stockpile to accept tallow and dye
{tallow}{permitdye}
```

You can see a more complex example of this with the `meltables` stockpiles in the lower left corner of the industry level.

The services level: handling multi-level dig blueprints



Services is a multi-level blueprint that includes a well cistern beneath the main level. Unwanted ramps caused by channeling are an annoyance, but we can avoid getting a ramp at the bottom of the cistern with careful use of *dig priorities*.

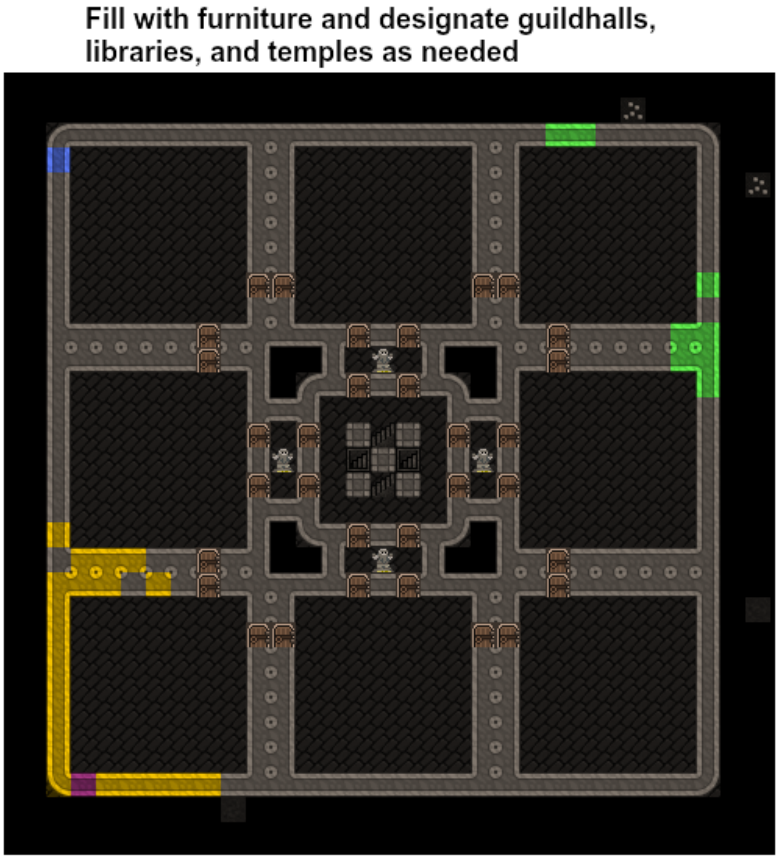
Tip

Use dig priorities to control ramp creation.

We can ensure the bottom level is carved out before the layer above is channelled by assigning the channel designations lower priorities (the `h5s` in the third layer – scroll down).

An alternative is to have a follow-up blueprint that removes any undesired ramps. We did this on the surface and farming levels with the miasma vents since it would be too complicated to synchronize the digging between the two layers.

The guildhall level: avoiding smoothing issues



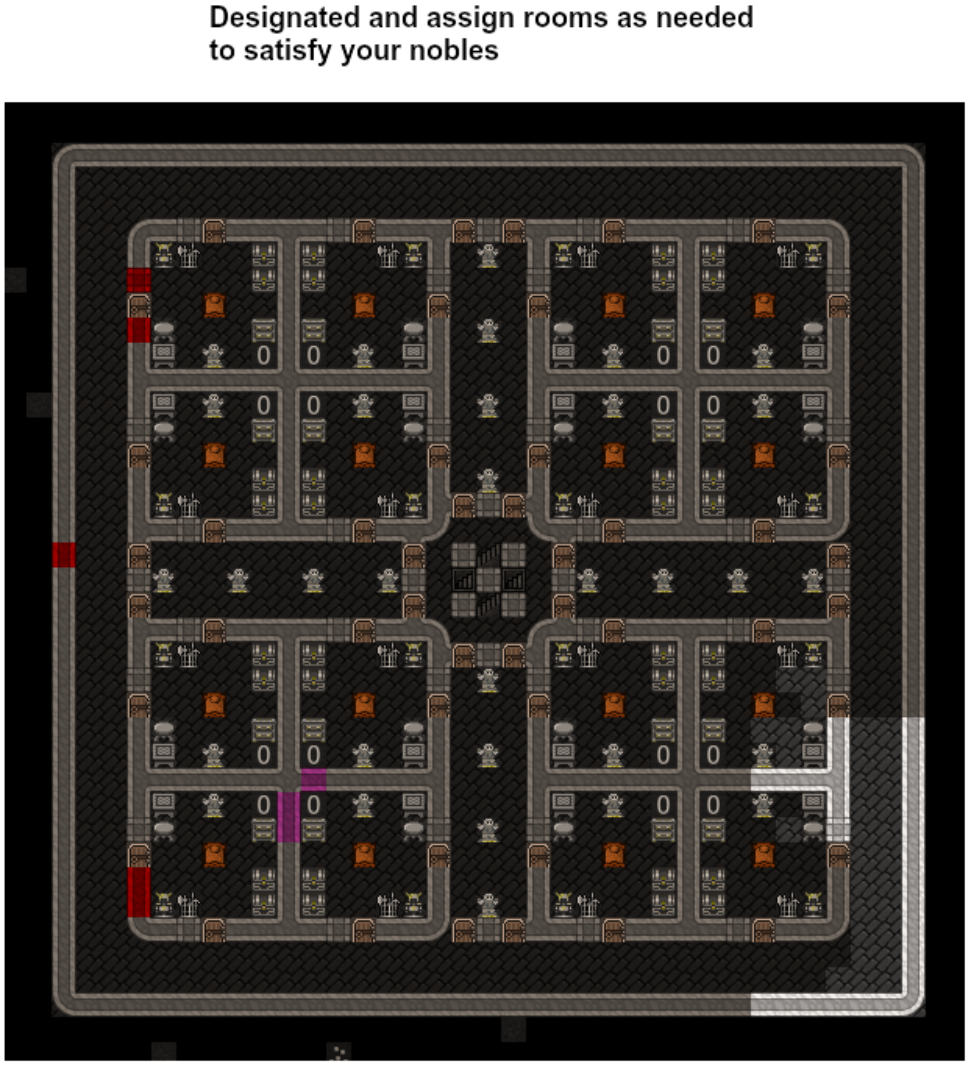
The goal of this level is to provide rooms for `locations` like guildhalls, libraries, and temples. The value of these rooms is very important, so we are likely to smooth and engrave everything. To smooth or engrave a wall tile, a dwarf has to be adjacent to it, and since some furniture, like statues, block dwarves from entering a tile, where you put them affects what you can access.

Tip

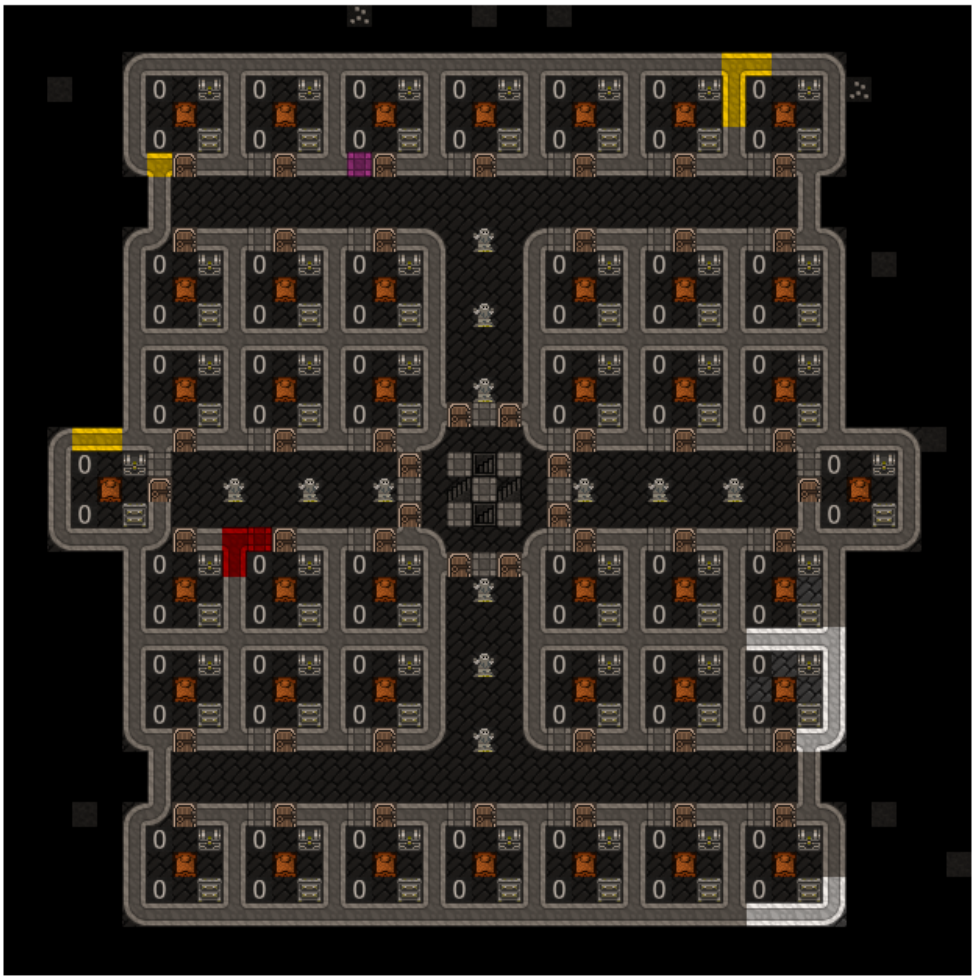
Don't put statues in corners unless you want to smooth everything first.

In the guildhall level, the statues are placed so as not to block any wall corners. This gives the player freedom for choosing when to smooth. If a statue blocks a corner, or if a line of statues blocks a wall segment, it forces the player to smooth before building the statues. Otherwise they have to bother with temporarily removing statues to smooth the walls behind them.

The beds levels: multi level meta blueprints



Spacious apartments make happy dwarves!
Doubles as a tomb. 40 rooms/80 urns per level.



The suites and apartments blueprints are straightforward. The only fancy bit is the meta blueprint that digs the stack of apartment levels, which brings us to our final tip:

Tip

Use meta blueprints to lay out repeated adjacent levels.

We couldn't use this technique for the entire fortress since there is often an aquifer between the farming and industry levels, and we can't know beforehand how many z-levels we need to skip. We can, however, automate the digging of everything from the industry level down, including designating all apartment levels at once. See the #meta blueprint in the Dreamfort help spreadsheet for how it uses a `repeat()` marker for the `/apartments1` blueprint to apply it to five z-levels at once.

That's it! I hope this guide was useful to you. Please leave feedback on the forums if you have ideas on how this guide

(or the dreamfort blueprints) can be improved!

7.4.9 Links

Quickfort links:

- *Quickfort command reference*
- *Quickfort Keystroke Alias Guide*
- *Blueprint Library Index*
- Quickfort forum thread
- DFHack issue tracker
- Blueprint library source
- Quickfort source code

Related tools:

- DFHack's *blueprint plugin* can generate blueprints from actual DF maps.
- DFHack's *buildingplan plugin* sets material and quality constraints for quickfort-placed buildings.
- Python Quickfort is the previous, Python-based implementation that DFHack's quickfort script was inspired by.

CHAPTER 8

About DFHack

These pages contain information about the general DFHack project.

8.1 Changelog

This file contains changes grouped by the stable release in which they first appeared. See *Building the changelogs* for more information.

See *Development Changelog* for a list of changes grouped by development releases.

Contents

8.1.1 DFHack 0.47.05-r5

New Plugins

- *spectate*: "spectator mode" – automatically follows dwarves doing things in your fort

New Scripts

- *devel/eventful-client*: useful for testing eventful events

New Tweaks

- *tweak*: `partial-items` displays percentage remaining for partially-consumed items such as hospital cloth

Fixes

- *autofarm*: removed restriction on only planting "discovered" plants
- *cxxrandom*: fixed exception when calling `bool_distribution`
- ***devel/query***:
  - fixed a problem printing parents when the starting path had lua pattern special characters in it
  - fixed a crash when trying to iterate over linked lists
- *gui/advfort*: encrust and stud jobs no longer consume reagents without actually improving the target item
- *luasocket*: return correct status code when closing socket connections so clients can know when to retry
- *quickfort*: contructions and bridges are now properly placed over natural ramps
- *setfps*: keep internal ratio of processing FPS to graphics FPS in sync when updating FPS

Misc Improvements

- ***autochop***:
  - only designate the amount of trees required to reach `max_logs`
  - preferably designate larger trees over smaller ones
- ***autonick***:
  - now displays help instead of modifying dwarf nicknames when run without parameters. use `autonick all` to rename all dwarves.

  - added `--quiet` and `--help` options
- ***blueprint*:**
  - `track` phase renamed to `carve`
  - carved fortifications and (optionally) engravings are now captured in generated blueprints
- *cursecheck*: new option, `--ids` prints creature and race IDs of the cursed creature
- ***debug*:**
  - DFHack log messages now have configurable headers (e.g. timestamp, origin plugin name, etc.) via the `debugfilter` command of the *debug* plugin
  - script execution log messages (e.g. "Loading script: dfhack_extras.init" can now be controlled with the `debugfilter` command. To hide the messages, add this line to your `dfhack.init` file: `debugfilter set Warning core script`
- ***DFHack Example Configuration File Index*:**
  - add mugs to `basic` manager orders
  - `onMapLoad_dreamfort.init` remove "cheaty" commands and new tweaks that are now in the default `dfhack.init-example` file
- *dig-now*: handle fortification carving
- ***Events from EventManager*:**
  - add new event type `JOB_STARTED`, triggered when a job first gains a worker
  - add new event type `UNIT_NEW_ACTIVE`, triggered when a new unit appears on the active list
- *gui/blueprint*: support new *blueprint* options and phases
- *gui/create-item*: Added "(chain)" annotation text for armours with the [CHAIN_METAL_TEXT] flag set
- *manipulator*: tweak colors to make the cursor easier to locate
- ***quickfort*:**
  - support transformations for blueprints that use expansion syntax
  - adjust direction affinity when transforming buildings (e.g. bridges that open to the north now open to the south when rotated 180 degrees)
  - automatically adjust cursor movements on the map screen in `#query` and `#config` modes when the blueprint is transformed. e.g. `{Up}` will be played back as `{Right}` when the blueprint is rotated clockwise and the direction key would move the map cursor
  - new blueprint mode: `#config`; for playing back key sequences that don't involve the map cursor (like configuring hotkeys, changing standing orders, or modifying military uniforms)
  - API function `apply_blueprint` can now take `data` parameters that are simple strings instead of coordinate maps. This allows easier application of blueprints that are just one cell.
- *stocks*: allow search terms to match the full item label, even when the label is truncated for length
- *tweak*: `stable-cursor` now keeps the cursor stable even when the viewport moves a small amount
- `dfhack.init-example`: recently-added tweaks added to example `dfhack.init` file

API

- add functions reverse-engineered from ambushing unit code: `Units::isHidden()`, `Units::isFortControlled()`, `Units::getOuterContainerRef()`, `Items::getOuterContainerRef()`

Lua

- *custom-raw-tokens*: library for accessing tokens added to raws by mods
- `dfhack.units`: Lua wrappers for functions reverse-engineered from ambushing unit code: `isHidden(unit)`, `isFortControlled(unit)`, `getOuterContainerRef(unit)`, `getOuterContainerRef(item)`
- `dialogs`: `show*` functions now return a reference to the created dialog
- `dwarfmode.enterSidebarMode()`: passing `df.ui_sidebar_mode.DesignateMine` now always results in you entering `DesignateMine` mode and not `DesignateChopTrees`, even when you looking at the surface (where the default designation mode is `DesignateChopTrees`)
- **`dwarfmode.MenuOverlay`:**
  - if `sidebar_mode` attribute is set, automatically manage entering a specific sidebar mode on show and restoring the previous sidebar mode on dismiss
  - new class function `renderMapOverlay` to assist with painting tiles over the visible map
- `ensure_key`: new global function for retrieving or dynamically creating Lua table mappings
- `safe_index`: now properly handles lua sparse tables that are indexed by numbers
- `string`: new function `escape_pattern()` escapes regex special characters within a string
- **`widgets`:**
  - unset values in `frame_inset` table default to `0`
  - `FilteredList` class now allows all punctuation to be typed into the filter and can match search keys that start with punctuation
  - minimum height of `ListBox` dialog is now calculated correctly when there are no items in the list (e.g. when a filter doesn't match anything)
  - if `autoarrange_subviews` is set, `Panels` will now automatically lay out widgets vertically according to their current height. This allows you to have widgets dynamically change height or become visible/hidden and you don't have to worry about recalculating frame layouts
  - new class `ResizingPanel` (subclass of `Panel`) automatically recalculates its own frame height based on the size, position, and visibility of its subviews
  - new class `HotkeyLabel` (subclass of `Label`) that displays and reacts to hotkeys
  - new class `CycleHotkeyLabel` (subclass of `Label`) allows users to cycle through a list of options by pressing a hotkey
  - new class `ToggleHotkeyLabel` (subclass of `CycleHotkeyLabel`) toggles between `On` and `Off` states
  - new class `WrappedLabel` (subclass of `Label`) provides autowrapping of text
  - new class `TooltipLabel` (subclass of `WrappedLabel`) provides tooltip-like behavior

Structures

- `adventure_optionst`: add missing `getUnitContainer` vmethod
- `historical_figure.T_skills`: add `account_balance` field
- `job`: add `improvement` field (union with `hist_figure_id` and `race`)
- `report_init.flags`: rename `sparring` flag to `hostile_combat`
- `viewscreen_loadgamest`: add missing `LoadingImageSets` and `LoadingDivinationSets` enum values to `cur_step` field

Documentation

- add more examples to the plugin example skeleton files so they are more informative for a newbie
- update download link and installation instructions for Visual C++ 2015 build tools on Windows
- update information regarding obtaining a compatible Windows build environment
- *confirm*: correct the command name in the plugin help text
- *cxxrandom*: added usage examples
- *String class extentions*: document DFHack string extensions (`startswith()`, `endswith()`, `split()`, `trim()`, `wrap()`, and `escape_pattern()`)
- *Quickfort Blueprint Editing Guide*: added screenshots to the Dreamfort case study and overall clarified text
- *Client libraries*: add new Rust client library
- `Lua API.rst`: added `isHidden(unit)`, `isFortControlled(unit)`, `getOuterContainerRef(unit)`, `getOuterContainerRef(item)`

8.1.2 DFHack 0.47.05-r4

Fixes

- ***blueprint***:
  - fixed passing incorrect parameters to *gui/blueprint* when you run `blueprint gui` with optional params
  - key sequences for constructed walls and down stairs are now correct
- *exportlegends*: fix issue where birth year was outputted as birth seconds
- ***quickfort***:
  - produce a useful error message instead of a code error when a bad query blueprint key sequence leaves the game in a mode that does not have an active cursor
  - restore functionality to the `--verbose` commandline flag
  - don't designate tiles for digging if they are within the bounds of a planned or constructed building
  - allow grates, bars, and hatches to be built on flat floor (like DF itself allows)
  - allow tracks to be built on hard, natural rock ramps
  - allow dig priority to be properly set for track designations

  - fix incorrect directions for tracks that extend south or east from a track segment pair specified with expansion syntax (e.g. T(4x4))
  - fix parsing of multi-part extended zone configs (e.g. when you set custom supply limits for hospital zones AND set custom flags for a pond)
  - fix error when attempting to set a custom limit for plaster powder in a hospital zone
- *tailor*: fixed some inconsistencies (and possible crashes) when parsing certain subcommands, e.g. `tailor help`
- *tiletypes-here*, *tiletypes-here-point*: fix crash when running from an unsuspended core context

Misc Improvements

- Core: DFHack now prints the name of the init script it is running to the console and stderr
- *automaterial*: ensure construction tiles are laid down in order when using *buildingplan* to plan the constructions
- ***blueprint***:
  - all blueprint phases are now written to a single file, using *quickfort* multi-blueprint file syntax. to get the old behavior of each phase in its own file, pass the `--splitby=phase` parameter to `blueprint`
  - you can now specify the position where the cursor should be when the blueprint is played back with *quickfort* by passing the `--playback-start` parameter
  - generated blueprints now have labels so *quickfort* can address them by name
  - all building types are now supported
  - multi-type stockpiles are now supported
  - non-rectangular stockpiles and buildings are now supported
  - blueprints are no longer generated for phases that have nothing to do (unless those phases are explicitly enabled on the commandline or gui)
  - new "track" phase that discovers and records carved tracks
  - new "zone" phase that discovers and records activity zones, including custom configuration for ponds, gathering, and hospitals
- *dig-now*: no longer leaves behind a designated tile when a tile was designated beneath a tile designated for channeling
- ***gui/blueprint***:
  - support the new `--splitby` and `--format` options for *blueprint*
  - hide help text when the screen is too short to display it
- *orders*: added `list` subcommand to show existing exported orders
- *Blueprint Library Index*: added light aquifer tap and pump stack blueprints (with step-by-step usage guides) to the quickfort blueprint library
- ***quickfort***:
  - Dreamfort blueprint set improvements: added iron and flux stock level indicators on the industry level and a prisoner processing quantum stockpile in the surface barracks. also added help text for how to manage sieges and how to manage prisoners after a siege.
  - add `quickfort.apply_blueprint()` API function that can be called directly by other scripts

  - by default, don't designate tiles for digging that have masterwork engravings on them. quality level to preserve is configurable with the new `--preserve-engravings` param
  - implement single-tile track aliases so engraved tracks can be specified tile-by-tile just like constructed tracks
  - allow blueprints to jump up or down multiple z-levels with a single command (e.g. `#>5` goes down 5 levels)
  - blueprints can now be repeated up and down a specified number of z-levels via `repeat` markers in meta blueprints or the `--repeat` commandline option
  - blueprints can now be rotated, flipped, and shifted via `transform` and `shift` markers in meta blueprints or the corresponding commandline options
- *quickfort*, *DFHack Example Configuration File Index*: Dreamfort blueprint set improvements based on playtesting and feedback. includes updated profession definitions.

Removed

- *digfort*: please use *quickfort* instead
- *fortplan*: please use *quickfort* instead

API

- `Buildings::findCivzonesAt()`: lookups now complete in constant time instead of linearly scanning through all civzones in the game
- `Job::remove_postings()`: use the job cancel vmethod graciously provided by The Toady One in place of a synthetic method derived from reverse engineering

Lua

- `argparse.processArgsGetopt()`: you can now have long form parameters that are not an alias for a short form parameter. For example, you can now have a parameter like `--longparam` without needing to have an equivalent one-letter `-l` param.
- `dwarfmode.enterSidebarMode()`: `df.ui_sidebar_mode.DesignateMine` is now a suported target sidebar mode

Structures

- `historical_figure_info.spheres`: give spheres vector a usable name
- `unit.enemy`: fix definition of `enemy_status_slot` and add `combat_side_id`

8.1.3 DFHack 0.47.05-r3

New Plugins

- *dig-now*: instantly completes dig designations (including smoothing and carving tracks)

New Scripts

- *autonick*: gives dwarves unique nicknames
- *build-now*: instantly completes planned building constructions
- *do-job-now*: makes a job involving current selection high priority
- *prioritize*: automatically boosts the priority of current and/or future jobs of specified types, such as hauling food, tanning hides, or pulling levers
- *reveal-adv-map*: exposes/hides all world map tiles in adventure mode

Fixes

- Core: `alt` keydown state is now cleared when DF loses and regains focus, ensuring the `alt` modifier state is not stuck on for systems that don't send standard keyup events in response to `alt-tab` window manager events
- Lua: `memscan.field_offset()`: fixed an issue causing *devel/export-dt-ini* to crash sometimes, especially on Windows
- *autofarm*: autofarm will now count plant growths as well as plants toward its thresholds
- *autogems*: no longer assigns gem cutting jobs to workshops with gem cutting prohibited in the workshop profile
- *devel/export-dt-ini*: fixed incorrect vtable address on Windows
- ***quickfort***:
  - allow machines (e.g. screw pumps) to be built on ramps just like DF allows
  - fix error message when the requested label is not found in the blueprint file

Misc Improvements

- *assign-beliefs*, *assign-facets*: now update needs of units that were changed
- *buildingplan*: now displays which items are attached and which items are still missing for planned buildings
- ***devel/query***:
  - updated script to v3.2 (i.e. major rewrite for maintainability/readability)
  - merged options `-query` and `-querykeys` into `-search`
  - merged options `-depth` and `-keydepth` into `-maxdepth`
  - replaced option `-safer` with `-excludetypes` and `-excludekinds`
  - improved how tile data is dealt with identification, iteration, and searching
  - added option `-findvalue`
  - added option `-showpaths` to print full data paths instead of nested fields
  - added option `-nopointers` to disable printing values with memory addresses
  - added option `-alignto` to set the value column's alignment
  - added options `-oneline` and alias `-1` to avoid using two lines for fields with metadata
  - added support for matching multiple patterns
  - added support for selecting the highlighted job, plant, building, and map block data

  - added support for selecting a Lua script (e.g. *dorf_tables*)
  - added support for selecting a Json file (e.g. dwarf_profiles.json)
  - removed options `-listall`, `-listfields`, and `-listkeys` - these are now simply default behaviour
  - `-table` now accepts the same abbreviations (global names, `unit`, `screen`, etc.) as *lua* and *gui/gm-editor*
- *dorf_tables*: integrated *devel/query* to show the table definitions when requested with `-list`
- *geld*: fixed `-help` option
- *gui/gm-editor*: made search case-insensitive
- ***orders*:**
  - support importing and exporting reaction-specific item conditions, like "lye-containing" for soap production orders
  - new `sort` command. sorts orders according to their repeat frequency. this prevents daily orders from blocking other orders for simlar items from ever getting completed.
- ***quickfort*:**
  - Dreamfort blueprint set improvements: extensive revision based on playtesting and feedback. includes updated `onMapLoad_dreamfort.init` settings file, enhanced automation orders, and premade profession definitions. see full changelog at https://github.com/DFHack/dfhack/pull/1921 and https://github.com/DFHack/dfhack/pull/1925
  - accept multiple commands, list numbers, and/or blueprint lables on a single commandline
- *tailor*: allow user to specify which materials to be used, and in what order
- *tiletypes-here*, *tiletypes-here-point*: add `--cursor` and `--quiet` options to support non-interactive use cases
- *unretire-anyone*: replaced the 'undead' descriptor with 'reanimated' to make it more mod-friendly
- *warn-starving*: added an option to only check sane dwarves

API

- The `Items` module `moveTo*` and `remove` functions now handle projectiles

Internals

- Install tests in the scripts repo into hack/scripts/test/scripts when the CMake variable BUILD_TESTS is defined

Lua

- new global function: `safe_pairs(iterable[, iterator_fn])` will iterate over the `iterable` (a table or iterable userdata) with the `iterator_fn` (`pairs` if not otherwise specified) if iteration is possible. If iteration is not possible or would throw an error, for example if `nil` is passed as the `iterable`, the iteration is just silently skipped.

Structures

- `cursed_tomb`: new struct type
- `job_item`: identified several fields
- `ocean_wave_maker`: new struct type
- `worldgen_parms`: moved to new struct type

Documentation

- *DFHack Example Configuration File Index*: documentation for all of *Dreamfort*'s supporting files (useful for all forts, not just Dreamfort!)
- *Blueprint Library Index*: updated dreamfort documentation and added screenshots

8.1.4 DFHack 0.47.05-r2

New Scripts

- *clear-webs*: removes all webs on the map and/or frees any webbed creatures
- *devel/block-borders*: overlay that displays map block borders
- *devel/luacov*: generate code test coverage reports for script development. Define the `DFHACK_ENABLE_LUACOV=1` environment variable to start gathering coverage metrics.
- *fix/drop-webs*: causes floating webs to fall to the ground
- *gui/blueprint*: interactive frontend for the *blueprint* plugin (with mouse support!)
- *gui/mass-remove*: mass removal/suspension tool for buildings and constructions
- *reveal-hidden-sites*: exposes all undiscovered sites
- *set-timeskip-duration*: changes the duration of the "Updating World" process preceding the start of a new game, enabling you to jump in earlier or later than usual

Fixes

- Fixed an issue preventing some external scripts from creating zones and other abstract buildings (see note about room definitions under "Internals")
- Fixed an issue where scrollable text in Lua-based screens could prevent other widgets from scrolling
- ***bodyswap*:**
  - stopped prior party members from tagging along after bodyswapping and reloading the map
  - made companions of bodyswapping targets get added to the adventurer party - they can now be viewed using the in-game party system
- ***buildingplan*:**
  - fixed an issue where planned constructions designated with DF's sizing keys (`umkh`) would sometimes be larger than requested
  - fixed an issue preventing other plugins like *automaterial* from planning constructions if the "enable all" buildingplan setting was turned on

  - made navigation keys work properly in the materials selection screen when alternate keybindings are used

- *color-schemes*: fixed an error in the `register` subcommand when the DF path contains certain punctuation characters
- *command-prompt*: fixed issues where overlays created by running certain commands (e.g. *gui/liquids*, *gui/teleport*) would not update the parent screen correctly
- *dwarfvet*: fixed a crash that could occur with hospitals overlapping with other buildings in certain ways
- *embark-assistant*: fixed faulty early exit in first search attempt when searching for waterfalls
- *gui/advfort*: fixed an issue where starting a workshop job while not standing at the center of the workshop required advancing time manually
- *gui/unit-info-viewer*: fixed size description displaying unrelated values instead of size
- *orders*: fixed crash when importing orders with malformed IDs
- ***quickfort***:
  - comments in blueprint cells no longer prevent the rest of the row from being read. A cell with a single '#' marker in it, though, will still stop the parser from reading further in the row.
  - fixed an off-by-one line number accounting in blueprints with implicit `#dig` modelines
  - changed to properly detect and report an error on sub-alias params with no values instead of just failing to apply the alias later (if you really want an empty value, use `{Empty}` instead)
  - improved handling of non-rectangular and non-solid extent-based structures (like fancy-shaped stockpiles and farm plots)
  - fixed conversion of numbers to DF keycodes in `#query` blueprints
  - fixed various errors with cropping across the map edge
  - properly reset config to default values in `quickfort reset` even if if the `dfhack-config/quickfort/quickfort.txt` config file doesn't mention all config vars. Also now works even if the config file doesn't exist.
- stonesense: fixed a crash that could occur when ctrl+scrolling or closing the Stonesense window
- `quickfortress.csv` blueprint: fixed refuse stockpile config and prevented stockpiles from covering stairways

Misc Improvements

- Added adjectives to item selection dialogs, used in tools like *gui/create-item* - this makes it possible to differentiate between different types of high/low boots, shields, etc. (some of which are procedurally generated)
- ***blueprint***:
  - made `depth` and `name` parameters optional. `depth` now defaults to `1` (current level only) and `name` defaults to "blueprint"
  - `depth` can now be negative, which will result in the blueprints being written from the highest z-level to the lowest. Before, blueprints were always written from the lowest z-level to the highest.
  - added the `--cursor` option to set the starting coordinate for the generated blueprints. A game cursor is no longer necessary if this option is used.
- *devel/annc-monitor*: added `report enable|disable` subcommand to filter combat reports
- *embark-assistant*: slightly improved performance of surveying and improved code a little

- *gui/advfort*: added workshop name to workshop UI
- ***quickfort*:**
  - the Dreamfort blueprint set can now be comfortably built in a 1x1 embark
  - added the `--cursor` option for running a blueprint at specific coordinates instead of starting at the game cursor position
  - added more helpful error messages for invalid modeline markers
  - added support for extra space characters in blueprints
  - added a warning when an invalid alias is encountered instead of silently ignoring it
  - made more quiet when the `--quiet` parameter is specified
- *setfps*: improved error handling
- stonesense: sped up startup time
- *tweak* hide-priority: changed so that priorities stay hidden (or visible) when exiting and re-entering the designations menu
- *unretire-anyone*: the historical figure selection list now includes the `SYN_NAME` (necromancer, vampire, etc) of figures where applicable

API

- Added `dfhack.maps.getPlantAtTile(x, y, z)` and `dfhack.maps.getPlantAtTile(pos)`, and updated `dfhack.gui.getSelectedPlant()` to use it
- Added `dfhack.units.teleport(unit, pos)`

Internals

- Room definitions and extents are now created for abstract buildings so callers don't have to initialize the room structure themselves
- The DFHack test harness is now much easier to use for iterative development. Configuration can now be specified on the commandline, there are more test filter options, and the test harness can now easily rerun tests that have been run before.
- The `test/main` command to invoke the test harness has been renamed to just `test`
- Unit tests can now use `delay_until(predicate_fn, timeout_frames)` to delay until a condition is met
- Unit tests must now match any output expected to be printed via `dfhack.printerr()`
- Unit tests now support fortress mode (allowing tests that require a fortress map to be loaded) - note that these tests are skipped by continuous integration for now, pending a suitable test fortress

Lua

- new library: `argparse` is a collection of commandline argument processing functions
- **new string utility functions:**
  - `string:wrap(width)` wraps a string at space-separated word boundaries
  - `string:trim()` removes whitespace characters from the beginning and end of the string

  - `string:split(delimiter, plain)` splits a string with the given delimiter and returns a table of substrings. if `plain` is specified and set to `true`, `delimiter` is interpreted as a literal string instead of as a pattern (the default)

- new utility function: `utils.normalizePath()`: normalizes directory slashes across platoforms to `/` and coaleses adjacent directory separators
- *reveal*: now exposes `unhideFlood(pos)` functionality to Lua
- *xlsxreader*: added Lua class wrappers for the xlsxreader plugin API
- **`argparse.processArgsGetopt()` (previously `utils.processArgsGetopt()`):**
  - now returns negative numbers (e.g. `-10`) in the list of positional parameters instead of treating it as an option string equivalent to `-1 -0`
  - now properly handles `--` like GNU `getopt` as a marker to treat all further parameters as non-options
  - now detects when required arguments to long-form options are missing
- `gui.dwarfmode`: new function: `enterSidebarMode(sidebar_mode, max_esc)` which uses keypresses to get into the specified sidebar mode from whatever the current screen is
- `gui.Painter`: fixed error when calling `viewport()` method

Structures

- Identified remaining rhythm beat enum values
- `ui_advmode.interactions`: identified some fields related to party members
- `ui_advmode_menu`: identified several enum items
- **`ui_advmode`:**
  - identified several fields
  - renamed `wait` to `rest_mode` and changed to an enum with correct values
- `viewscreen_legendsst.cur_page`: added missing `Books` enum item, which fixes some other values

Documentation

- Added more client library implementations to the *remote interface docs*

8.1.5 DFHack 0.47.05-r1

Fixes

- *confirm*: stopped exposing alternate names when convicting units
- *embark-assistant*: fixed bug in soil depth determination for ocean tiles
- *orders*: don't crash when importing orders with malformed JSON
- *prospect*: improved pre embark rough estimates, particularly for small clusters
- *quickfort*: raw numeric *Dig priorities* (e.g. `3`, which is a valid shorthand for `d3`) now works when used in .xlsx blueprints

Misc Improvements

- *autohauler*: allowed the `Alchemist` labor to be enabled in *manipulator* and other labor screens so it can be used for its intended purpose of flagging that no hauling labors should be assigned to a dwarf. Before, the only way to set the flag was to use an external program like Dwarf Therapist.
- *embark-assistant*: slightly improved performance of surveying
- *gui/no-dfhack-init*: clarified how to dismiss dialog that displays when no `dfhack.init` file is found
- ***quickfort***:
  - Dreamfort blueprint set improvements: significant refinements across the entire blueprint set. Dreamfort is now much faster, much more efficient, and much easier to use. The checklist now includes a mini-walkthrough for quick reference. The spreadsheet now also includes embark profile suggestions
  - added aliases for configuring masterwork and artifact core quality for all stockpile categories that have them; made it possible to take from multiple stockpiles in the `quantumstop` alias
  - an active cursor is no longer required for running #notes blueprints (like the dreamfort walkthrough)
  - you can now be in any mode with an active cursor when running `#query` blueprints (before you could only be in a few "approved" modes, like look, query, or place)
  - refined `#query` blueprint sanity checks: cursor should still be on target tile at end of configuration, and it's ok for the screen ID to change if you are destroying (or canceling destruction of) a building
  - now reports how many work orders were added when generating manager orders from blueprints in the gui dialog
  - added `--dry-run` option to process blueprints but not change any game state
  - you can now specify the number of desired barrels, bins, and wheelbarrows for individual stockpiles when placing them
  - `quickfort orders` on a `#place` blueprint will now enqueue manager orders for barrels, bins, or wheelbarrows that are explicitly set in the blueprint.
  - you can now add alias definitions directly to your blueprint files instead of having to put them in a separate aliases.txt file. makes sharing blueprints with custom alias definitions much easier.
  - new commandline options for setting the initial state of the gui dialog. for example: `quickfort gui -l dreamfort notes` will start the dialog filtered for the dreamfort walkthrough blueprints

Structures

- Dropped support for 0.47.03-0.47.04
- Identified scattered enum values (some rhythm beats, a couple of corruption unit thoughts, and a few language name categories)
- `viewscreen_loadgamest`: renamed `cur_step` enumeration to match style of `viewscreen_adopt_regionst` and `viewscreen_savegamest`
- `viewscreen_savegamest`: identified `cur_step` enumeration

Documentation

- *digfort*: added deprecation warnings - digfort has been replaced by *quickfort*
- *fortplan*: added deprecation warnings - fortplan has been replaced by *quickfort*

8.1.6 DFHack 0.47.04-r5

New Scripts

- *gui/quickfort*: fast access to the quickfort interactive dialog
- *workorder-recheck*: resets the selected work order to the `Checking` state

Fixes

- ***embark-assistant***:
  - fixed order of factors when calculating min temperature
  - improved performance of surveying
- ***quickfort***:
  - fixed eventual crashes when creating zones
  - fixed library aliases for tallow and iron, copper, and steel weapons
  - zones are now created in the active state by default
  - solve rare crash when changing UI modes
- *search*: fixed crash when searching the `k` sidebar and navigating to another tile with certain keys, like < or >
- *seedwatch*: fixed an issue where the plugin would disable itself on map load
- *stockflow*: fixed `j` character being intercepted when naming stockpiles
- *stockpiles*: no longer outputs hotkey help text beneath *stockflow* hotkey help text

Misc Improvements

- Lua label widgets (used in all standard message boxes) are now scrollable with Up/Down/PgUp/PgDn keys
- *autofarm*: now fallows farms if all plants have reached the desired count
- ***buildingplan***:
  - added ability to set global settings from the console, e.g. `buildingplan set boulders false`
  - added "enable all" option for buildingplan (so you don't have to enable all building types individually). This setting is not persisted (just like quickfort_mode is not persisted), but it can be set from onMapLoad.init
  - modified `Planning Mode` status in the UI to show whether the plugin is in quickfort mode, "enable all" mode, or whether just the building type is enabled.
- ***quickfort***:
  - Dreamfort blueprint set improvements: added a streamlined checklist for all required dreamfort commands and gave names to stockpiles, levers, bridges, and zones
  - added aliases for bronze weapons and armor
  - added alias for tradeable crafts
  - new blueprint mode: `#ignore`, useful for scratch space or personal notes

  - implement `{Empty}` keycode for use in quickfort aliases; useful for defining blank-by-default alias values
  - more flexible commandline parsing allowing for more natural parameter ordering (e.g. where you used to have to write `quickfort list dreamfort -l` you can now write `quickfort list -l dreamfort`)
  - print out blueprint names that a `#meta` blueprint is applying so it's easier to understand what meta blueprints are doing
  - whitespace is now allowed between a marker name and the opening parenthesis in blueprint modelines. for example, `#dig start (5; 5)` is now valid (you used to be required to write `#dig start(5; 5)`)

Lua

- `dfhack.run_command()`: changed to interface directly with the console when possible, which allows interactive commands and commands that detect the console encoding to work properly
- `processArgsGetopt()` added to utils.lua, providing a callback interface for parameter parsing and getopt-like flexibility for parameter ordering and combination (see docs in `library/lua/utils.lua` and `library/lua/3rdparty/alt_getopt.lua` for details).

Structures

- `job`: identified `order_id` field

Documentation

- Added documentation for Lua's `dfhack.run_command()` and variants

8.1.7 DFHack 0.47.04-r4

New Scripts

- *fix/corrupt-equipment*: fixes some military equipment-related corruption issues that can cause DF crashes

Fixes

- Fixed an issue on some Linux systems where DFHack installed through a package manager would attempt to write files to a non-writable folder (notably when running *exportlegends* or *gui/autogems*)
- *adaptation*: fixed handling of units with no cave adaptation suffered yet
- *assign-goals*: fixed error preventing new goals from being created
- *assign-preferences*: fixed handling of preferences for flour
- ***buildingplan*:**
  - fixed an issue preventing artifacts from being matched when the maximum item quality is set to `artifacts`
  - stopped erroneously matching items to buildings while the game is paused
  - fixed a crash when pressing 0 while having a noble room selected

- *deathcause*: fixed an error when inspecting certain corpses
- *dwarfmonitor*: fixed a crash when opening the `prefs` screen if units have vague preferences
- *dwarfvet*: fixed a crash that could occur when discharging patients
- ***embark-assistant*:**
  - fixed an issue causing incursion resource matching (e.g. sand/clay) to skip some tiles if those resources were provided only through incursions
  - corrected river size determination by performing it at the MLT level rather than the world tile level
- ***quickfort*:**
  - fixed handling of modifier keys (e.g. `{Ctrl}` or `{Alt}`) in query blueprints
  - fixed misconfiguration of nest boxes, hives, and slabs that were preventing them from being built from build blueprints
  - fixed valid placement detection for floor hatches, floor grates, and floor bars (they were erroneously being rejected from open spaces and staircase tops)
  - fixed query blueprint statistics being added to the wrong metric when both a query and a zone blueprint are run by the same meta blueprint
  - added missing blueprint labels in gui dialog list
  - fixed occupancy settings for extent-based structures so that stockpiles can be placed within other stockpiles (e.g. in a checkerboard or bullseye pattern)
- *search*: fixed an issue where search options might not display if screens were destroyed and recreated programmatically (e.g. with *quickfort*)
- *unsuspend*: now leaves buildingplan-managed buildings alone and doesn't unsuspend underwater tasks
- *workflow*: fixed an error when creating constraints on "mill plants" jobs and some other plant-related jobs
- *zone*: fixed an issue causing the `enumnick` subcommand to run when attempting to run `assign`, `unassign`, or `slaughter`

Misc Improvements

- ***buildingplan*:**
  - added support for all buildings, furniture, and constructions (except for instruments)
  - added support for respecting building job_item filters when matching items, so you can set your own programmatic filters for buildings before submitting them to buildingplan
  - changed default filter setting for max quality from `artifact` to `masterwork`
  - changed min quality adjustment hotkeys from 'qw' to 'QW' to avoid conflict with existing hotkeys for setting roller speed - also changed max quality adjustment hotkeys from 'QW' to 'AS' to make room for the min quality hotkey changes
  - added a new global settings page accessible via the `G` hotkey when on any building build screen; `Quickfort Mode` toggle for legacy Python Quickfort has been moved to this page
  - added new global settings for whether generic building materials should match blocks, boulders, logs, and/or bars - defaults are everything but bars
- *devel/export-dt-ini*: updated for Dwarf Therapist 41.2.0
- *embark-assistant*: split the lair types displayed on the local map into mound, burrow, and lair

- *gui/advfort*: added support for linking to hatches and pressure plates with mechanisms
- *modtools/add-syndrome*: added support for specifying syndrome IDs instead of names
- *probe*: added more output for designations and tile occupancy
- ***quickfort***:
  - The Dreamfort sample blueprints now have complete walkthroughs for each fort level and importable orders that automate basic fort stock management
  - added more blueprints to the blueprints library: several bedroom layouts, the Saracen Crypts, and the complete fortress example from Python Quickfort: TheQuickFortress
  - query blueprint aliases can now accept parameters for dynamic expansion - see dfhack-config/quickfort/aliases.txt for details
  - alias names can now include dashes and underscores (in addition to letters and numbers)
  - improved speed of first call to `quickfort list` significantly, especially for large blueprint libraries
  - added `query_unsafe` setting to disable query blueprint error checking - useful for query blueprints that send unusual key sequences
  - added support for bookcases, display cases, and offering places (altars)
  - added configuration support for zone pit/pond, gather, and hospital sub-menus in zone blueprints
  - removed `buildings_use_blocks` setting and replaced it with more flexible functionality in *buildingplan*
  - added support for creating uninitialized stockpiles with `c`

API

- *buildingplan*: added Lua interface API
- `Buildings::setSize()`: changed to reuse existing extents when possible
- `dfhack.job.isSuitableMaterial()`: added an item type parameter so the `non_economic` flag can be properly handled (it was being matched for all item types instead of just boulders)

Lua

- `utils.addressof()`: fixed for raw userdata

Structures

- `building_extents_type`: new enum, used for `building_extents.extents`
- `world_mountain_peak`: new struct (was previously inline) - used in `world_data.mountain_peaks`

Documentation

- *Quickfort Keystroke Alias Guide*: alias syntax and alias standard library documentation for *quickfort* blueprints
- *Blueprint Library Index*: overview of the quickfort blueprint library

8.1.8 DFHack 0.47.04-r3

New Plugins

- *xlsxreader*: provides an API for Lua scripts to read Excel spreadsheets

New Scripts

- *quickfort*: DFHack-native implementation of quickfort with many new features and integrations - see the *Quickfort Blueprint Editing Guide* for details
- *timestream*: controls the speed of the calendar and creatures
- *uniform-unstick*: prompts units to reevaluate their uniform, by removing/dropping potentially conflicting worn items

Fixes

- *ban-cooking*: fixed an error in several subcommands
- *buildingplan*: fixed handling of buildings that require buckets
- *getplants*: fixed a crash that could occur on some maps
- *search*: fixed an issue causing item counts on the trade screen to display inconsistently when searching
- ***stockpiles*:**
  - fixed a crash when loading food stockpiles
  - fixed an error when saving furniture stockpiles

Misc Improvements

- ***createitem*:**
  - added support for plant growths (fruit, berries, leaves, etc.)
  - added an `inspect` subcommand to print the item and material tokens of existing items, which can be used to create additional matching items
- *embark-assistant*: added support for searching for taller waterfalls (up to 50 z-levels tall)
- *search*: added support for searching for names containing non-ASCII characters using their ASCII equivalents
- *stocks*: added support for searching for items containing non-ASCII characters using their ASCII equivalents
- *unretire-anyone*: made undead creature names appear in the historical figure list
- ***zone*:**
  - added an `enumnick` subcommand to assign enumerated nicknames (e.g "Hen 1", "Hen 2"...)
  - added slaughter indication to `uinfo` output

API

- Added `DFHack::to_search_normalized()` (Lua: `dfhack.toSearchNormalized()`) to convert non-ASCII alphabetic characters to their ASCII equivalents

Structures

- `history_event_masterpiece_createdst`: fixed alignment, including subclasses, and identified `skill_at_time`
- `item_body_component`: fixed some alignment issues and identified some fields (also applies to subclasses like `item_corpsest`)
- `stockpile_settings`: removed `furniture.sand_bags` (no longer present)

Documentation

- Fixed syntax highlighting of most code blocks to use the appropriate language (or no language) instead of Python

8.1.9 DFHack 0.47.04-r2

New Scripts

- *animal-control*: helps manage the butchery and gelding of animals
- *devel/kill-hf*: kills a historical figure
- *geld*: gelds or ungelds animals
- *list-agreements*: lists all guildhall and temple agreements
- *list-waves*: displays migration wave information for citizens/units
- *ungeld*: ungelds animals (wrapper around *geld*)

New Tweaks

- *tweak* do-job-now: adds a job priority toggle to the jobs list
- *tweak* reaction-gloves: adds an option to make reactions produce gloves in sets with correct handedness

Fixes

- Fixed a segfault when attempting to start a headless session with a graphical PRINT_MODE setting
- Fixed an issue with the macOS launcher failing to un-quarantine some files
- Fixed `Units::isEggLayer`, `Units::isGrazer`, `Units::isMilkable`, `Units::isTrainableHunting`, `Units::isTrainableWar`, and `Units::isTamable` ignoring the unit's caste
- Linux: fixed `dfhack.getDFPath()` (Lua) and `Process::getPath()` (C++) to always return the DF root path, even if the working directory has changed
- ***digfort*:**
  - fixed y-line tracking when .csv files contain lines with only commas
  - fixed an issue causing blueprints touching the southern or eastern edges of the map to be rejected (northern and western edges were already allowed). This allows blueprints that span the entire embark area.
- *embark-assistant*: fixed a couple of incursion handling bugs.

- *embark-skills*: fixed an issue with structures causing the `points` option to do nothing
- ***exportlegends*:**
  - fixed an issue where two different `<reason>` tags could be included in a `<historical_event>`
  - stopped including some tags with `-1` values which don't provide useful information
- *getplants*: fixed issues causing plants to be collected even if they have no growths (or unripe growths)
- *gui/advfort*: fixed "operate pump" job
- *gui/load-screen*: fixed an issue causing longer timezones to be cut off
- ***labormanager*:**
  - fixed handling of new jobs in 0.47
  - fixed an issue preventing custom furnaces from being built
- ***modtools/moddable-gods*:**
  - fixed an error when creating the historical figure
  - removed unused `-domain` and `-description` arguments
  - made `-depictedAs` argument work
- ***names*:**
  - fixed an error preventing the script from working
  - fixed an issue causing renamed units to display their old name in legends mode and some other places
- *pref-adjust*: fixed some compatibility issues and a potential crash
- ***remotefortressreader*:**
  - fixed a couple crashes that could result from decoding invalid enum items (`site_realization_building_type` and `improvement_type`)
  - fixed an issue that could cause block coordinates to be incorrect
- *rendermax*: fixed a hang that could occur when enabling some renderers, notably on Linux
- **stonesense:**
  - fixed a crash when launching Stonesense
  - fixed some issues that could cause the splash screen to hang

Misc Improvements

- Linux/macOS: Added console keybindings for deleting words (Alt+Backspace and Alt+d in most terminals)
- ***add-recipe*:**
  - added tool recipes (minecarts, wheelbarrows, stepladders, etc.)
  - added a command explanation or error message when entering an invalid command
- *armoks-blessing*: added adjustments to values and needs
- ***blueprint*:**
  - now writes blueprints to the `blueprints/` subfolder instead of the df root folder

  - now automatically creates folder trees when organizing blueprints into subfolders (e.g. `blueprint 30 30 1 rooms/dining dig` will create the file `blueprints/rooms/dining-dig.csv`); previously it would fail if the `blueprints/rooms/` directory didn't already exist

- *confirm*: added a confirmation dialog for convicting dwarves of crimes
- *devel/query*: added many new query options
- ***digfort***:
  - handled double quotes (") at the start of a string, allowing .csv files exported from spreadsheets to work without manual modification
  - documented that removing ramps, cutting trees, and gathering plants are indeed supported
  - added a `force` option to truncate blueprints if the full blueprint would extend off the edge of the map
- ***dwarf-op***:
  - added ability to select dwarves based on migration wave
  - added ability to protect dwarves based on symbols in their custom professions
- ***exportlegends***:
  - changed some flags to be represented by self-closing tags instead of true/false strings (e.g. `<is_volcano/>`) - note that this may require changes to other XML-parsing utilities
  - changed some enum values from numbers to their string representations
  - added ability to save all files to a subfolder, named after the region folder and date by default
- *gui/advfort*: added support for specifying the entity used to determine available resources
- *gui/gm-editor*: added support for automatically following ref-targets when pressing the `i` key
- *manipulator*: added a new column option to display units' goals
- *modtools/moddable-gods*: added support for `neuter` gender
- ***pref-adjust***:
  - added support for adjusting just the selected dwarf
  - added a new `goth` profile
- *remove-stress*: added a `-value` argument to enable setting stress level directly
- *workorder*: changed default frequency from "Daily" to "OneTime"

API

- Added `Filesystem::mkdir_recursive`
- Extended `Filesystem::listdir_recursive` to optionally make returned filenames relative to the start directory
- `Units`: added goal-related functions: `getGoalType()`, `getGoalName()`, `isGoalAchieved()`

Internals

- Added support for splitting scripts into multiple files in the `scripts/internal` folder without polluting the output of *ls*

Lua

- Added a `ref_target` field to primitive field references, corresponding to the `ref-target` XML attribute
- Made `dfhack.units.getRaceNameById()`, `dfhack.units.getRaceBabyNameById()`, and `dfhack.units.getRaceChildNameById()` available to Lua

Ruby

- Updated `item_find` and `building_find` to use centralized logic that works on more screens

Structures

- Added a new `<df-other-vectors-type>`, which allows `world.*.other` collections of vectors to use the correct subtypes for items
- `creature_raw`: renamed `gender` to `sex` to match the field in `unit`, which is more frequently used
- `crime`: identified `witnesses`, which contains the data held by the old field named `reports`
- `intrigue`: new type (split out from `historical_figure_relationships`)
- `items_other_id`: removed `BAD`, and by extension, `world.items.other.BAD`, which was overlapping with `world.items.bad`
- `job_type`: added job types new to 0.47
- `plant_raw`: material_defs now contains arrays rather than loose fields
- `pronoun_type`: new enum (previously documented in field comments)
- `setup_character_info`: fixed a couple alignment issues (needed by *embark-skills*)
- `ui_advmode_menu`: identified some new enum items

Documentation

- Added some new dev-facing pages, including dedicated pages about the remote API, memory research, and documentation
- Expanded the installation guide
- Made a couple theme adjustments

8.1.10 DFHack 0.47.04-r1

New Scripts

- *color-schemes*: manages color schemes
- *devel/print-event*: prints the description of an event by ID or index
- *devel/sc*: checks size of structures
- *devel/visualize-structure*: displays the raw memory of a structure
- *gui/color-schemes*: an in-game interface for *color-schemes*
- *light-aquifers-only*: changes heavy aquifers to light aquifers

- *on-new-fortress*: runs DFHack commands only in a new fortress
- *once-per-save*: runs DFHack commands unless already run in the current save
- *resurrect-adv*: brings your adventurer back to life
- *reveal-hidden-units*: exposes all sneaking units
- *workorder*: allows queuing manager jobs; smart about shear and milk creature jobs

Fixes

- Fixed a crash in `find()` for some types when no world is loaded
- Fixed a crash when starting DFHack in headless mode with no terminal
- Fixed translation of certain types of in-game names
- *autogems*: fixed an issue with binned gems being ignored in linked stockpiles
- *catsplosion*: fixed error when handling races with only one caste (e.g. harpies)
- ***deep-embark***:
  - prevented running in non-fortress modes
  - ensured that only the newest wagon is deconstructed
- *devel/visualize-structure*: fixed padding detection for globals
- ***exportlegends***:
  - added UTF-8 encoding and XML escaping for more fields
  - added checking for unhandled structures to avoid generating invalid XML
  - fixed missing fields in `history_event_assume_identityst` export
- ***full-heal***:
  - fixed issues with removing corpses
  - fixed resurrection for non-historical figures
  - when resurrected by specifying a corpse, units now appear at the location of the corpse rather than their location of death
  - resurrected units now have their tile occupancy set (and are placed in the prone position to facilitate this)
- *spawnunit*: fixed an error when forwarding some arguments but not a location to *modtools/create-unit*
- *stocks*: fixed display of book titles
- *teleport*: fixed setting new tile occupancy
- *tweak* embark-profile-name: fixed handling of the native shift+space key

Misc Improvements

- Added "bit" suffix to downloads (e.g. 64-bit)
- **Tests:**
  - moved from DF folder to hack/scripts folder, and disabled installation by default

  - made test runner script more flexible
- ***deep-embark***:
  - improved support for using directly from the DFHack console
  - added a `-clear` option to cancel
- *devel/export-dt-ini*: updated some field names for DT for 0.47
- *devel/visualize-structure*: added human-readable lengths to containers
- *dfhack-run*: added color output support
- ***embark-assistant***:
  - updated embark aquifer info to show all aquifer kinds present
  - added neighbor display, including kobolds (SKULKING) and necro tower count
  - updated aquifer search criteria to handle the new variation
  - added search criteria for embark initial tree cover
  - added search criteria for necro tower count, neighbor civ count, and specific neighbors. Should handle additional entities, but not tested
- ***exportlegends***:
  - added identity information
  - added creature raw names and flags
  - made interaction export more robust and human-readable
  - removed empty `<item_subtype>` and `<claims>` tags
  - added evilness and force IDs to regions
  - added profession and weapon info to relevant entities
  - added support for many new history events in 0.47
  - added historical event relationships and supplementary data
- ***full-heal***:
  - made resurrection produce a historical event viewable in Legends mode
  - made error messages more explanatory
- *getplants*: added switches for designations for farming seeds and for max number designated per plant
- *gui/prerelease-warning*: updated links and information about nightly builds
- *install-info*: added DFHack build ID to report
- *manipulator*: added intrigue to displayed skills
- *modtools/create-item*: added `-matchingGloves` and `-matchingShoes` arguments
- ***modtools/create-unit***:
  - added `-equip` option to equip created units
  - added `-skills` option to give skills to units
  - added `-profession` and `-customProfession` options to adjust unit professions
  - added `-duration` argument to make the unit vanish after some time

  - added `-locationRange` argument to allow spawning in a random position within a defined area
  - added `-locationType` argument to specify the type of location to spawn in
- *modtools/syndrome-trigger*: enabled simultaneous use of `-synclass` and `-syndrome`
- *repeat*: added `-list` option
- *search*: added support for the fortress mode justice screen
- `dfhack.init-example`: enabled *autodump*

API

- Added `Items::getBookTitle` to get titles of books. Catches titles buried in improvements, unlike getDescription.

Internals

- Added separate changelogs in the scripts and df-structures repos
- Improved support for tagged unions, allowing tools to access union fields more safely
- Moved `reversing` scripts to df_misc repo

Lua

- `pairs()` now returns available class methods for DF types

Structures

- Added an XML schema for validating df-structures syntax
- Added globals: `cur_rain`, `cur_rain_counter`, `cur_snow`, `cur_snow_counter`, `weathertimer`, `jobvalue`, `jobvalue_setter`, `interactitem`, `interactinvslot`, `handleannounce`, `preserveannounce`, `updatelightstate`
- Added `divination_set_next_id` and `image_set_next_id` globals
- Dropped support for 0.44.12-0.47.02
- `abstract_building_type`: added types (and subclasses) new to 0.47
- `activity_entry_type`: new enum type
- `adventure_optionst`: identified many vmethods
- `agreement_details_data_plot_sabotage`: new struct type, along with related `agreement_details_type.PlotSabotage`
- `agreement_details_type`: added enum
- **`agreement_details`:**
  - added struct type (and many associated data types)
  - identified most fields of most sub-structs
- `agreement_party`: added struct type
- `announcement_type`: added types new to 0.47

- `architectural_element`: new enum
- `artifact_claim_type`: added enum
- **`artifact_claim`:**
  - added struct type
  - identified several fields
- `artifact_record`: identified several fields
- `battlefield`: new struct type
- `breath_attack_type`: added `SHARP_ROCK`
- `breed`: new struct type
- `building_offering_placest`: new class
- `building_type`: added `OfferingPlace`
- **`caste_raw_flags`:**
  - renamed many items to match DF names
  - renamed and identified many flags to match information from Toady
- `creature_handler`: identified vmethods
- `creature_interaction_effect`: added subclasses new to 0.47
- **`creature_raw_flags`:**
  - identified several more items
  - renamed many items to match DF names
  - renamed and identified many flags to match information from Toady
- `crime_type`: new enum type
- `crime`: removed fields of `reports` that are no longer present
- `dance_form`: identified most fields
- `dfhack_room_quality_level`: added enum attributes for names of rooms of each quality
- `d_init`: added settings new to 0.47
- `entity_name_type`: added `MERCHANT_COMPANY`, `CRAFT_GUILD`
- `entity_position_responsibility`: added values new to 0.47
- `entity_site_link_type`: new enum type
- `export_map_type`: new enum type
- `fortress_type`: added enum
- `general_ref_type`: added `UNIT_INTERROGATEE`
- `ghost_type`: added `None` value
- `goal_type`: added goals types new to 0.47
- `histfig_site_link`: added subclasses new to 0.47
- `historical_entity.flags`: identified several flags
- `historical_entity.relations`: renamed from `unknown1b` and identified several fields

- historical_figure.vague_relationships: identified
- historical_figure_info.known_info: renamed from secret, identified some fields
- historical_figure: renamed unit_id2 to nemesis_id
- history_event_circumstance_info: new struct type (and changed several history_event subclasses to use this)
- history_event_collection: added subtypes new to 0.47
- **history_event_context:**
  - added lots of new fields
  - identified fields
- history_event_reason_info: new struct type (and changed several history_event subclasses to use this)
- **history_event_reason:**
  - added captions for all items
  - added items new to 0.47
- history_event_type: added types for events new to 0.47, as well as corresponding history_event subclasses (too many to list here)
- **honors_type:**
  - added struct type
  - identified several fields
- identity_type: new enum
- identity: renamed civ to entity_id, identified type
- image_set: new struct type
- interaction_effect_create_itemst: new struct type
- interaction_effect_summon_unitst: new struct type
- interaction_effect: added subtypes new to 0.47
- interaction_source_experimentst: added class type
- interaction_source_usage_hint: added values new to 0.47
- interface_key: added items for keys new to 0.47
- interrogation_report: new struct type
- itemdef_flags: new enum, with GENERATED flag
- item: identified several vmethods
- job_skill: added INTRIGUE, RIDING
- justification: new enum
- lair_type: added enum
- layer_type: new enum type
- lever_target_type: identified LeverMechanism and TargetMechanism values
- monument_type: added enum

- musical_form: identified fields, including some renames. Also identified fields in scale and rhythm
- next_global_id: added enum
- plant.damage_flags: added is_dead
- plot_role_type: new enum type
- plot_strategy_type: new enum type
- poetic_form_action: added Beseech
- region_weather: new struct type
- relationship_event_supplement: new struct type
- relationship_event: new struct type
- setup_character_info: expanded significantly in 0.47
- specific_ref: moved union data to data field
- squad_order_cause_trouble_for_entityst: identified fields
- text_system: added layout for struct
- tile_occupancy: added varied_heavy_aquifer
- tool_uses: added items: PLACE_OFFERING, DIVINATION, GAMES_OF_CHANCE
- ui_look_list: moved union fields to data and renamed to match type enum
- ui_sidebar_menus.location: added new profession-related fields, renamed and fixed types of deity-related fields
- ui_sidebar_mode: added ZonesLocationInfo
- unit_action: rearranged as tagged union with new sub-types; existing code should be compatible
- unit_thought_type: added several new thought types
- vague_relationship_type: new enum type
- vermin_flags: identified is_roaming_colony
- viewscreen_counterintelligencest: new class (only layout identified so far)
- viewscreen_justicest: identified interrogation-related fields
- viewscreen_workquota_detailsst: identified fields
- world_data.field_battles: identified and named several fields

8.1.11 DFHack 0.44.12-r3

New Plugins

- *autoclothing*: automatically manage clothing work orders
- *autofarm*: replaces the previous Ruby script of the same name, with some fixes
- *map-render*: allows programmatically rendering sections of the map that are off-screen
- *tailor*: automatically manages keeping your dorfs clothed

New Scripts

- *assign-attributes*: changes the attributes of a unit
- *assign-beliefs*: changes the beliefs of a unit
- *assign-facets*: changes the facets (traits) of a unit
- *assign-goals*: changes the goals of a unit
- *assign-preferences*: changes the preferences of a unit
- *assign-profile*: sets a dwarf's characteristics according to a predefined profile
- *assign-skills*: changes the skills of a unit
- *combat-harden*: sets a unit's combat-hardened value to a given percent
- *deep-embark*: allows embarking underground
- *devel/find-twbt*: finds a TWBT-related offset needed by the new *map-render* plugin
- *dwarf-op*: optimizes dwarves for fort-mode work; makes managing labors easier
- *forget-dead-body*: removes emotions associated with seeing a dead body
- *gui/create-tree*: creates a tree at the selected tile
- *linger*: takes over your killer in adventure mode
- *modtools/create-tree*: creates a tree
- *modtools/pref-edit*: add, remove, or edit the preferences of a unit
- *modtools/set-belief*: changes the beliefs (values) of units
- *modtools/set-need*: sets and edits unit needs
- *modtools/set-personality*: changes the personality of units
- *modtools/spawn-liquid*: spawns water or lava at the specified coordinates
- *set-orientation*: edits a unit's orientation
- *unretire-anyone*: turns any historical figure into a playable adventurer

Fixes

- Fixed a crash in the macOS/Linux console when the prompt was wider than the screen width
- Fixed inconsistent results from `Units::isGay` for asexual units
- Fixed some cases where Lua filtered lists would not properly intercept keys, potentially triggering other actions on the same screen
- ***autofarm*:**
  - fixed biome detection to properly determine crop assignments on surface farms
  - reimplemented as a C++ plugin to make proper biome detection possible
- *bodyswap*: fixed companion list not being updated often enough
- *cxxrandom*: removed some extraneous debug information
- *digfort*: now accounts for z-level changes when calculating maximum y dimension
- ***embark-assistant*:**

  - fixed bug causing crash on worlds without generated metals (as well as pruning vectors as originally intended).
  - fixed bug causing mineral matching to fail to cut off at the magma sea, reporting presence of things that aren't (like DF does currently).
  - fixed bug causing half of the river tiles not to be recognized.
  - added logic to detect some river tiles DF doesn't generate data for (but are definitely present).
- *eventful*: fixed invalid building ID in some building events
- *exportlegends*: now escapes special characters in names properly
- *getplants*: fixed designation of plants out of season (note that picked plants are still designated incorrectly)
- *gui/autogems*: fixed error when no world is loaded
- ***gui/companion-order*:**
  - fixed error when resetting group leaders
  - `leave` now properly removes companion links
- *gui/create-item*: fixed module support - can now be used from other scripts
- ***gui/stamper*:**
  - stopped "invert" from resetting the designation type
  - switched to using DF's designation keybindings instead of custom bindings
  - fixed some typos and text overlapping
- ***modtools/create-unit*:**
  - fixed an error associating historical entities with units
  - stopped recalculating health to avoid newly-created citizens triggering a "recover wounded" job
  - fixed units created in arena mode having blank names
  - fixed units created in arena mode having the wrong race and/or interaction effects applied after creating units manually in-game
  - stopped units from spawning with extra items or skills previously selected in the arena
  - stopped setting some unneeded flags that could result in glowing creature tiles
  - set units created in adventure mode to have no family, instead of being related to the first creature in the world
- ***modtools/reaction-product-trigger*:**
  - fixed an error dealing with reactions in adventure mode
  - blocked `\\BUILDING_ID` for adventure mode reactions
  - fixed `-clear` to work without passing other unneeded arguments
- ***modtools/reaction-trigger*:**
  - fixed a bug when determining whether a command was run
  - fixed handling of `-resetPolicy`
- *mousequery*: fixed calculation of map dimensions, which was sometimes preventing scrolling the map with the mouse when TWBT was enabled
- *remotefortressreader*: fixed a crash when a unit's path has a length of 0

- stonesense: fixed crash due to wagons and other soul-less creatures
- *tame*: now sets the civ ID of tamed animals (fixes compatibility with *autobutcher*)
- *title-folder*: silenced error when `PRINT_MODE` is set to `TEXT`

Misc Improvements

- Added a note to *dfhack-run* when called with no arguments (which is usually unintentional)
- On macOS, the launcher now attempts to un-quarantine the rest of DFHack
- *bodyswap*: added arena mode support
- *combine-drinks*: added more default output, similar to *combine-plants*
- *createitem*: added a list of valid castes to the "invalid caste" error message, for convenience
- *devel/export-dt-ini*: added more size information needed by newer Dwarf Therapist versions
- *dwarfmonitor*: enabled widgets to access other scripts and plugins by switching to the core Lua context
- ***embark-assistant*:**
  - added an in-game option to activate on the embark screen
  - changed waterfall detection to look for level drop rather than just presence
  - changed matching to take incursions, i.e. parts of other biomes, into consideration when evaluating tiles. This allows for e.g. finding multiple biomes on single tile embarks.
  - changed overlay display to show when incursion surveying is incomplete
  - changed overlay display to show evil weather
  - added optional parameter "fileresult" for crude external harness automated match support
  - improved focus movement logic to go to only required world tiles, increasing speed of subsequent searches considerably
- *exportlegends*: added rivers to custom XML export
- *exterminate*: added support for a special `enemy` caste
- ***gui/gm-unit*:**
  - added support for editing:
  - added attribute editor
  - added orientation editor
  - added editor for bodies and body parts
  - added color editor
  - added belief editor
  - added personality editor
- *modtools/create-item*: documented already-existing `-quality` option
- ***modtools/create-unit*:**
  - added the ability to specify `\\LOCAL` for the fort group entity
  - now enables the default labours for adult units with CAN_LEARN.
  - now sets historical figure orientation.

  - improved speed of creating multiple units at once
  - made the script usable as a module (from other scripts)
- ***modtools/reaction-trigger*:**
  - added `-ignoreWorker`: ignores the worker when selecting the targets
  - changed the default behavior to skip inactive/dead units; added `-dontSkipInactive` to include creatures that are inactive
  - added `-range`: controls how far elligible targets can be from the workshop
  - syndromes now are applied before commands are run, not after
  - if both a command and a syndrome are given, the command only runs if the syndrome could be applied
- *mousequery*: made it more clear when features are enabled
- ***remotefortressreader*:**
  - added a basic framework for controlling and reading the menus in DF (currently only supports the building menu)
  - added support for reading item raws
  - added a check for whether or not the game is currently saving or loading, for utilities to check if it's safe to read from DF
  - added unit facing direction estimate and position within tiles
  - added unit age
  - added unit wounds
  - added tree information
  - added check for units' current jobs when calculating the direction they are facing

API

- Added new `plugin_load_data` and `plugin_save_data` events for plugins to load/save persistent data
- Added `Maps::GetBiomeType` and `Maps::GetBiomeTypeByRef` to infer biome types properly
- Added `Units::getPhysicalDescription` (note that this depends on the `unit_get_physical_description` offset, which is not yet available for all DF builds)

Internals

- Added new Persistence module
- Cut down on internal DFHack dependencies to improve build times
- Improved concurrency in event and server handlers
- Persistent data is now stored in JSON files instead of historical figures - existing data will be migrated when saving
- stonesense: fixed some OpenGL build issues on Linux

Lua

- Exposed `gui.dwarfmode.get_movement_delta` and `gui.dwarfmode.get_hotkey_target`
- `dfhack.run_command` now returns the command's return code

Ruby

- Made `unit_ishostile` consistently return a boolean

Structures

- Added `unit_get_physical_description` function offset on some platforms
- **Added/identified types:**
  - `assume_identity_mode`
  - `musical_form_purpose`
  - `musical_form_style`
  - `musical_form_pitch_style`
  - `musical_form_feature`
  - `musical_form_vocals`
  - `musical_form_melodies`
  - `musical_form_interval`
  - `unit_emotion_memory`
- `need_type`: fixed `PrayOrMeditate` typo
- `personality_facet_type`, `value_type`: added `NONE` values
- `twbt_render_map`: added for 64-bit 0.44.12 (for *map-render*)

8.1.12 DFHack 0.44.12-r2

New Plugins

- *debug*: manages runtime debug print category filtering
- *nestboxes*: automatically scan for and forbid fertile eggs incubating in a nestbox

New Scripts

- *devel/query*: searches for field names in DF objects
- *extinguish*: puts out fires
- *tame*: sets tamed/trained status of animals

Fixes

- *building-hacks*: fixed error when dealing with custom animation tables
- *devel/test-perlin*: fixed Lua error (`math.pow()`)
- *embark-assistant*: fixed crash when entering finder with a 16x16 embark selected, and added 16 to dimension choices
- *embark-skills*: fixed missing `skill_points_remaining` field
- ***full-heal***:
  - stopped wagon resurrection
  - fixed a minor issue with post-resurrection hostility
- ***gui/companion-order***:
  - fixed issues with printing coordinates
  - fixed issues with move command
  - fixed cheat commands (and removed "Power up", which was broken)
- *gui/gm-editor*: fixed reinterpret cast (`r`)
- *gui/pathable*: fixed error when sidebar is hidden with `Tab`
- ***labormanager***:
  - stopped assigning labors to ineligible dwarves, pets, etc.
  - stopped assigning invalid labors
  - added support for crafting jobs that use pearl
  - fixed issues causing cleaning jobs to not be assigned
  - added support for disabling management of specific labors
- *prospect*: (also affected *embark-tools*) - fixed a crash when prospecting an unusable site (ocean, mountains, etc.) with a large default embark size in d_init.txt (e.g. 16x16)
- *siege-engine*: fixed a few Lua errors (`math.pow()`, `unit.relationship_ids`)
- *tweak*: fixed `hotkey-clear`

Misc Improvements

- *armoks-blessing*: improved documentation to list all available arguments
- ***devel/export-dt-ini***:
  - added viewscreen offsets for DT 40.1.2
  - added item base flags offset
  - added needs offsets
- ***embark-assistant***:
  - added match indicator display on the right ("World") map
  - changed 'c'ancel to abort find if it's under way and clear results if not, allowing use of partial surveys.
  - added Coal as a search criterion, as well as a coal indication as current embark selection info.

- ***full-heal*:**
  - added `-all`, `-all_civ` and `-all_citizens` arguments
  - added module support
  - now removes historical figure death dates and ghost data
- *growcrops*: added `all` argument to grow all crops
- *gui/load-screen*: improved documentation
- *labormanager*: now takes nature value into account when assigning jobs
- *open-legends*: added warning about risk of save corruption and improved related documentation
- *points*: added support when in `viewscreen_setupdwarfgamest` and improved error messages
- *siren*: removed break handling (relevant `misc_trait_type` was no longer used - see "Structures" section)

API

- **New debug features related to *debug* plugin:**
  - Classes (C++ only): `Signal<Signature, type_tag>`, `DebugCategory`, `DebugManager`
  - Macros: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERR`, `DBG_DECLARE`, `DBG_EXTERN`

Internals

- Added a usable unit test framework for basic tests, and a few basic tests
- Added `CMakeSettings.json` with intellisense support
- Changed `plugins/CMakeLists.custom.txt` to be ignored by git and created (if needed) at build time instead
- Core: various thread safety and memory management improvements
- Fixed CMake build dependencies for generated header files
- Fixed custom `CMAKE_CXX_FLAGS` not being passed to plugins
- Linux/macOS: changed recommended build backend from Make to Ninja (Make builds will be significantly slower now)

Lua

- `utils`: new `OrderedTable` class

Structures

- Win32: added missing vtables for `viewscreen_storesst` and `squad_order_rescue_hfst`
- `activity_event_performancest`: renamed poem as written_content_id
- `body_part_status`: identified `gelded`
- `dance_form`: named musical_form_id and musical_written_content_id
- `incident_sub6_performance.participants`: named performance_event and role_index

- **incident_sub6_performance:**
  - made performance_event an enum
  - named poetic_form_id, musical_form_id, and dance_form_id
- misc_trait_type: removed LikesOutdoors, Hardened, TimeSinceBreak, OnBreak (all unused by DF)
- musical_form_instruments: named minimum_required and maximum_permitted
- musical_form: named voices field
- plant_tree_info: identified extent_east, etc.
- plant_tree_tile: gave connection bits more meaningful names (e.g. connection_east instead of thick_branches_1)
- poetic_form: identified many fields and related enum/bitfield types
- setup_character_info: identified skill_points_remaining (for *embark-skills*)
- ui.main: identified fortress_site
- ui.squads: identified kill_rect_targets_scroll
- ui: fixed alignment of main and squads (fixes *tweak* hotkey-clear and DF-AI)
- **unit_action.attack:**
  - identified attack_skill
  - added lightly_tap and spar_report flags
- unit_flags3: identified marked_for_gelding
- unit_personality: identified stress_drain, stress_boost, likes_outdoors, combat_hardened
- unit_storage_status: newly identified type, stores noble holdings information (used in viewscreen_layer_noblelistst)
- unit_thought_type: added new expulsion thoughts from 0.44.12
- viewscreen_layer_arena_creaturest: identified item- and name-related fields
- viewscreen_layer_militaryst: identified equip.assigned.assigned_items
- viewscreen_layer_noblelistst: identified storage_status (see unit_storage_status type)
- **viewscreen_new_regionst:**
  - identified rejection_msg, raw_folder, load_world_params
  - changed many int8_t fields to bool
- viewscreen_setupadventurest: identified some nemesis and personality fields, and page.ChooseHistfig
- world_data: added mountain_peak_flags type, including is_volcano
- world_history: identified names and/or types of some fields
- world_site: identified names and/or types of some fields
- written_content: named poetic_form

8.1.13 DFHack 0.44.12-r1

Fixes

- Fixed displayed names (from `Units::getVisibleName`) for units with identities
- Fixed potential memory leak in `Screen::show()`
- Fixed special characters in *command-prompt* and other non-console in-game outputs on Linux/macOS (in tools using `df2console`)
- *command-prompt*: added support for commands that require a specific screen to be visible, e.g. *spotclean*
- *die*: fixed Windows crash in exit handling
- *dwarfmonitor*, *manipulator*: fixed stress cutoffs
- *fix/dead-units*: fixed script trying to use missing isDiplomat function
- *gui/workflow*: fixed advanced constraint menu for crafts
- *modtools/force*: fixed a bug where the help text would always be displayed and nothing useful would happen
- *ruby*: fixed calling conventions for vmethods that return strings (currently `enabler.GetKeyDisplay()`)
- *startdwarf*: fixed on 64-bit Linux
- stonesense: fixed `PLANT:DESERT_LIME:LEAF` typo

Misc Improvements

- **Console:**
  - added support for multibyte characters on Linux/macOS
  - made the console exit properly when an interactive command is active (*liquids*, *mode*, *tiletypes*)
- Linux: added automatic support for GCC sanitizers in `dfhack` script
- Made the `DFHACK_PORT` environment variable take priority over `remote-server.json`
- Reduced time for designation jobs from tools like *dig* to be assigned workers
- *dfhack-run*: added support for port specified in `remote-server.json`, to match DFHack's behavior
- *digfort*: added better map bounds checking
- ***embark-assistant*:**
  - Switched to standard scrolling keys, improved spacing slightly
  - Introduced scrolling of Finder search criteria, removing requirement for 46 lines to work properly (Help/Info still formatted for 46 lines).
  - Added Freezing search criterion, allowing searches for NA/Frozen/At_Least_Partial/Partial/At_Most_Partial/Never Freezing embarks.
- ***rejuvenate*:**
  - Added `-all` argument to apply to all citizens
  - Added `-force` to include units under 20 years old
  - Clarified documentation
- ***remove-stress*:**

  - added support for `-all` as an alternative to the existing `all` argument for consistency
  - sped up significantly
  - improved output/error messages
  - now removes tantrums, depression, and obliviousness
- *ruby*: sped up handling of onupdate events

API

- Added C++-style linked list interface for DF linked lists
- **Added to `Units` module:**
  - `getStressCategory(unit)`
  - `getStressCategoryRaw(level)`
  - `stress_cutoffs` (Lua: `getStressCutoffs()`)
- Added `Screen::Hide` to temporarily hide screens, like *command-prompt*
- Exposed `Screen::zoom()` to C++ (was Lua-only)
- New functions: `Units::isDiplomat(unit)`

Internals

- Added documentation for all RPC functions and a build-time check
- Added support for build IDs to development builds
- Changed default build architecture to 64-bit
- jsoncpp: updated to version 1.8.4 and switched to using a git submodule
- Use `dlsym(3)` to find vtables from libgraphics.so

Lua

- Added `printall_recurse` to print tables and DF references recursively. It can be also used with `^` from the *lua* interpreter.
- `gui.widgets`: `List:setChoices` clones `choices` for internal table changes

Structures

- Added support for automatically sizing arrays indexed with an enum
- Added `start_dwarf_count` on 64-bit Linux again and fixed scanning script
- Dropped 0.44.10 support
- Dropped 0.44.11 support
- Removed stale generated CSV files and DT layouts from pre-0.43.05
- `announcement_type`: new in 0.44.11: `NEW_HOLDING`, `NEW_MARKET_LINK`
- `army_controller`: added new vector from 0.44.11

- belief_system: new type, few fields identified
- breath_attack_type: added OTHER
- historical_figure_info.relationships.list: added unk_3a-unk_3c fields at end
- history_event_entity_expels_hfst: added (new in 0.44.11)
- history_event_site_surrenderedst: added (new in 0.44.11)
- history_event_type: added SITE_SURRENDERED, ENTITY_EXPELS_HF (new in 0.44.11)
- interface_key: added bindings new in 0.44.11
- mental_picture: new type, some fields identified
- **mission_report:**
  - new type (renamed, was mission before)
  - identified some fields
- mission: new type (used in viewscreen_civliststst)
- occupation_type: new in 0.44.11: MESSENGER
- profession: new in 0.44.11: MESSENGER
- spoils_report: new type, most fields identified
- syndrome: identified a few fields
- ui.squads: Added fields new in 0.44.12
- **ui_sidebar_menus:**
  - unit.in_squad: renamed to unit.squad_list_opened, fixed location
  - unit: added expel_error and other unknown fields new in 0.44.11
  - hospital: added, new in 0.44.11
  - num_speech_tokens, unk_17d8: moved out of command_line to fix layout on x64
- **viewscreen_civliststst:**
  - identified new pages
  - identified new messenger-related fields
  - fixed layout and identified many fields
- **viewscreen_image_creatorst:**
  - fixed layout
  - identified many fields
- viewscreen_locationsst: identified edit_input
- viewscreen_reportliststst: added new mission and spoils report-related fields (fixed layout)
- world.languages: identified (minimal information; whole languages stored elsewhere)
- **world.status:**
  - mission_reports: renamed, was missions
  - spoils_reports: identified
- world.unk_131ec0, world.unk_131ef0: researched layout

- `world.worldgen_status`: identified many fields
- `world`: `belief_systems`: identified

8.1.14 DFHack 0.44.10-r2

New Plugins

- *cxxrandom*: exposes some features of the C++11 random number library to Lua

New Scripts

- *add-recipe*: adds unknown crafting recipes to the player's civ
- *gui/stamper*: allows manipulation of designations by transforms such as translations, reflections, rotations, and inversion

Fixes

- Fixed many tools incorrectly using the `dead` unit flag (they should generally check `flags2.killed` instead)
- Fixed many tools passing incorrect arguments to printf-style functions, including a few possible crashes (*change-layer*, *follow*, *forceequip*, *generated-creature-renamer*)
- Fixed several bugs in Lua scripts found by static analysis (df-luacheck)
- Fixed `-g` flag (GDB) in Linux `dfhack` script (particularly on x64)
- *autochop*, *autodump*, *autogems*, *automelt*, *autotrade*, *buildingplan*, *dwarfmonitor*, *fix-unit-occupancy*, *fortplan*, *stockflow*: fix issues with periodic tasks not working for some time after save/load cycles
- ***autogems*:**
  - stop running repeatedly when paused
  - fixed crash when furnaces are linked to same stockpiles as jeweler's workshops
- *autogems*, *fix-unit-occupancy*: stopped running when a fort isn't loaded (e.g. while embarking)
- *autounsuspend*: now skips planned buildings
- *ban-cooking*: fixed errors introduced by kitchen structure changes in 0.44.10-r1
- *buildingplan*, *fortplan*: stopped running before a world has fully loaded
- *deramp*: fixed deramp to find designations that already have jobs posted
- *dig*: fixed "Inappropriate dig square" announcements if digging job has been posted
- *fixnaked*: fixed errors due to emotion changes in 0.44
- *remove-stress*: fixed an error when running on soul-less units (e.g. with `-all`)
- *reveal*: stopped revealing tiles adjacent to tiles above open space inappropriately
- *stockpiles*: `loadstock` now sets usable and unusable weapon and armor settings
- *stocks*: stopped listing carried items under stockpiles where they were picked up from

Misc Improvements

- Added script name to messages produced by `qerror()` in Lua scripts
- Fixed an issue in around 30 scripts that could prevent edits to the files (adding valid arguments) from taking effect
- Linux: Added several new options to `dfhack` script: `--remotegdb`, `--gdbserver`, `--strace`
- *bodyswap*: improved error handling
- *buildingplan*: added max quality setting
- *caravan*: documented (new in 0.44.10-alpha1)
- *deathcause*: added "slaughtered" to descriptions
- ***embark-assistant*:**
  - changed region interaction matching to search for evil rain, syndrome rain, and reanimation rather than interaction presence (misleadingly called evil weather), reanimation, and thralling
  - gave syndrome rain and reanimation wider ranges of criterion values
- *fix/dead-units*: added a delay of around 1 month before removing units
- *fix/retrieve-units*: now re-adds units to active list to counteract *fix/dead-units*
- *item-descriptions*: fixed several grammatical errors
- ***modtools/create-unit*:**
  - added quantity argument
  - now selects a caste at random if none is specified
- ***mousequery*:**
  - migrated several features from TWBT's fork
  - added ability to drag with left/right buttons
  - added depth display for TWBT (when multilevel is enabled)
  - made shift+click jump to lower levels visible with TWBT
- *title-version*: added version to options screen too

API

- **New functions (also exposed to Lua):**
  - `Units::isKilled()`
  - `Units::isActive()`
  - `Units::isGhost()`
- Removed Vermin module (unused and obsolete)

Internals

- Added build option to generate symbols for large generated files containing df-structures metadata
- Added fallback for YouCompleteMe database lookup failures (e.g. for newly-created files)
- Improved efficiency and error handling in `stl_vsprintf` and related functions
- jsoncpp: fixed constructor with `long` on Linux

Lua

- Added `profiler` module to measure lua performance
- Enabled shift+cursor movement in WorkshopOverlay-derived screens

Structures

- `incident_sub6_performance`: identified some fields
- `item_body_component`: fixed location of `corpse_flags`
- `job_handler`: fixed static array layout
- `job_type`: added `is_designation` attribute
- `unit_flags1`: renamed `dead` to `inactive` to better reflect its use
- `unit_personality`: fixed location of `current_focus` and `undistracted_focus`
- `unit_thought_type`: added `SawDeadBody` (new in 0.44.10)

8.1.15 DFHack 0.44.10-r1

New Scripts

- *bodyswap*: shifts player control over to another unit in adventure mode
- *caravan*: adjusts properties of caravans
- *devel/find-primitive*: finds a primitive variable in memory
- *gui/autogems*: a configuration UI for the *autogems* plugin

New Tweaks

- *tweak* kitchen-prefs-all: adds an option to toggle cook/brew for all visible items in kitchen preferences
- *tweak* stone-status-all: adds an option to toggle the economic status of all stones

Fixes

- Fixed uninitialized pointer being returned from `Gui::getAnyUnit()` in rare cases
- Lua: registered `dfhack.constructions.designateRemove()` correctly
- Units::getAnyUnit(): fixed a couple problematic conditions and potential segfaults if global addresses are missing

- *autohauler*, *autolabor*, *labormanager*: fixed fencepost error and potential crash
- *dwarfvet*: fixed infinite loop if an animal is not accepted at a hospital
- *exterminate*: fixed documentation of `this` option
- ***full-heal*:**
  - units no longer have a tendency to melt after being healed
  - healed units are no longer treated as patients by hospital staff
  - healed units no longer attempt to clean themselves unsuccessfully
  - wounded fliers now regain the ability to fly upon being healing
  - now heals suffocation, numbness, infection, spilled guts and gelding
- *liquids*: fixed "range" command to default to 1 for dimensions consistently
- ***modtools/create-unit*:**
  - creatures of the appropriate age are now spawned as babies or children where applicable
  - fix: civ_id is now properly assigned to historical_figure, resolving several hostility issues (spawned pets are no longer attacked by fortress military!)
  - fix: unnamed creatures are no longer spawned with a string of numbers as a first name
- *prospect*: fixed crash due to invalid vein materials
- *search*: fixed 4/6 keys in unit screen search
- *stockpiles*: stopped sidebar option from overlapping with *autodump*
- *tweak* block-labors: fixed two causes of crashes related in the v-p-l menu
- *tweak* max-wheelbarrow: fixed conflict with building renaming
- ***view-item-info*:**
  - stopped appending extra newlines permanently to descriptions
  - fixed an error with some armor

Misc Improvements

- Added logo to documentation
- Documented several missing `dfhack.gui` Lua functions
- *adv-rumors*: bound to Ctrl-A
- *autogems*: can now blacklist arbitrary gem types (see *gui/autogems*)
- *blueprint*: added a basic Lua API
- *command-prompt*: added support for `Gui::getSelectedPlant()`
- *devel/export-dt-ini*: added tool offsets for DT 40
- *devel/save-version*: added current DF version to output
- *exterminate*: added more words for current unit, removed warning
- *fpause*: now pauses worldgen as well
- *gui/advfort*: bound to Ctrl-T

- *gui/room-list*: added support for `Gui::getSelectedBuilding()`
- *gui/unit-info-viewer*: bound to Alt-I
- *install-info*: added information on tweaks
- *modtools/create-unit*: made functions available to other scripts
- ***search*:**
  - added support for stone restrictions screen (under `z`: Status)
  - added support for kitchen preferences (also under `z`)

API

- **New functions (all available to Lua as well):**
  - `Buildings::getRoomDescription()`
  - `Items::checkMandates()`
  - `Items::canTrade()`
  - `Items::canTradeWithContents()`
  - `Items::isRouteVehicle()`
  - `Items::isSquadEquipment()`
  - `Kitchen::addExclusion()`
  - `Kitchen::findExclusion()`
  - `Kitchen::removeExclusion()`
- syndrome-util: added `eraseSyndromeData()`

Internals

- Added function names to DFHack's NullPointer and InvalidArgument exceptions
- Added some build scripts for Sublime Text
- Added `Gui::inRenameBuilding()`
- Changed submodule URLs to relative URLs so that they can be cloned consistently over different protocols (e.g. SSH)
- Fixed compiler warnings on all supported build configurations
- Linux: required plugins to have symbols resolved at link time, for consistency with other platforms
- Windows build scripts now work with non-C system drives

Structures

- `dfhack_room_quality_level`: new enum
- `glowing_barrier`: identified `triggered`, added comments
- `item_flags2`: renamed `has_written_content` to `unk_book`
- `kitchen_exc_type`: new enum (for `ui.kitchen`)

- `mandate.mode`: now an enum
- `unit_personality.emotions.flags.memory`: identified
- `viewscreen_kitchenprefst.forbidden`, `possible`: now a bitfield, `kitchen_pref_flag`
- `world_data.feature_map`: added extensive documentation (in XML)

8.1.16 DFHack 0.44.09-r1

Fixes

- Fixed some CMake warnings (CMP0022)
- Support for building on Ubuntu 18.04
- *digtype*: stopped designating non-vein tiles (open space, trees, etc.)
- *embark-assistant*: fixed detection of reanimating biomes
- *fix/dead-units*: fixed a bug that could remove some arriving (not dead) units
- *labormanager*: fixed crash due to dig jobs targeting some unrevealed map blocks
- *modtools/item-trigger*: fixed token format in help text

Misc Improvements

- Reorganized changelogs and improved changelog editing process
- ***embark-assistant*:**
  - Added search for adamantine
  - Now supports saving/loading profiles
- *fillneeds*: added `-all` option to apply to all units
- ***modtools/item-trigger*:**
  - added support for multiple type/material/contaminant conditions
  - added the ability to specify inventory mode(s) to trigger on
- *remotefortressreader*: added flows, instruments, tool names, campfires, ocean waves, spiderwebs

Internals

- OS X: Can now build with GCC 7 (or older)

Structures

- Several new names in instrument raw structures
- `army`: added vector new in 0.44.07
- `building_type`: added human-readable `name` attribute
- `furnace_type`: added human-readable `name` attribute
- `identity`: identified `profession`, `civ`

- `manager_order_template`: fixed last field type
- `site_reputation_report`: named `reports` vector
- `viewscreen_createquotast`: fixed layout
- `workshop_type`: added human-readable `name` attribute
- `world.language`: moved `colors`, `shapes`, `patterns` to `world.descriptors`
- **`world.reactions`, `world.reaction_categories`: moved to new compound, `world.reactions`. Requires rena**
  - `world.reactions` to `world.reactions.reactions`
  - `world.reaction_categories` to `world.reactions.reaction_categories`

8.1.17 DFHack 0.44.05-r2

New Plugins

- *embark-assistant*: adds more information and features to embark screen

New Scripts

- *adv-fix-sleepers*: fixes units in adventure mode who refuse to wake up (Bug 6798)
- *hermit*: blocks caravans, migrants, diplomats (for hermit challenge)

New Features

- With `PRINT_MODE:TEXT`, setting the `DFHACK_HEADLESS` environment variable will hide DF's display and allow the console to be used normally. (Note that this is intended for testing and is not very useful for actual gameplay.)

Fixes

- *devel/export-dt-ini*: fix language_name offsets for DT 39.2+
- *devel/inject-raws*: fixed gloves and shoes (old typo causing errors)
- *remotefortressreader*: fixed an issue with not all engravings being included
- *view-item-info*: fixed an error with some shields

Misc Improvements

- *adv-rumors*: added more keywords, including names
- *autochop*: can now exclude trees that produce fruit, food, or cookable items
- *remotefortressreader*: added plant type support

8.1.18 DFHack 0.44.05-r1

New Scripts

- *break-dance*: Breaks up a stuck dance activity
- *devel/check-other-ids*: Checks the validity of "other" vectors in the `world` global
- *devel/dump-offsets*: prints an XML version of the global table included in in DF
- *fillneeds*: Use with a unit selected to make them focused and unstressed
- *firestarter*: Lights things on fire: items, locations, entire inventories even!
- *flashstep*: Teleports adventurer to cursor
- *ghostly*: Turns an adventurer into a ghost or back
- *gui/cp437-table*: An in-game CP437 table
- *questport*: Sends your adventurer to the location of your quest log cursor
- *view-unit-reports*: opens the reports screen with combat reports for the selected unit

Fixes

- Fixed a crash that could occur if a symbol table in symbols.xml had no content
- Fixed issues with the console output color affecting the prompt on Windows
- *autolabor*, *autohauler*, *labormanager*: added support for "put item on display" jobs and building/destroying display furniture
- *createitem*: stopped items from teleporting away in some forts
- ***devel/inject-raws*:**
  - now recognizes spaces in reaction names
  - now recognizes spaces in reaction names
- *dig*: added support for designation priorities - fixes issues with designations from `digv` and related commands having extremely high priority
- ***dwarfmonitor*:**
  - fixed display of creatures and poetic/music/dance forms on `prefs` screen
  - added "view unit" option
  - now exposes the selected unit to other tools
- *exportlegends*: fixed an error that could occur when exporting empty lists
- *gui/gm-editor*: fixed an error when editing primitives in Lua tables
- *gui/gm-unit*: can now edit mining skill
- *gui/quickcmd*: stopped error from adding too many commands
- *modtools/create-unit*: fixed error when domesticating units
- *names*: fixed many errors
- *quicksave*: fixed an issue where the "Saving…" indicator often wouldn't appear

Misc Improvements

- The console now provides suggestions for built-in commands
- *binpatch*: now reports errors for empty patch files
- *devel/export-dt-ini*: avoid hardcoding flags
- ***exportlegends*:**
  - reordered some tags to match DF's order
  - added progress indicators for exporting long lists
- *force*: now provides useful help
- ***full-heal*:**
  - can now select corpses to resurrect
  - now resets body part temperatures upon resurrection to prevent creatures from freezing/melting again
  - now resets units' vanish countdown to reverse effects of *exterminate*
- *gui/gm-editor*: added enum names to enum edit dialogs
- ***gui/gm-unit*:**
  - added a profession editor
  - misc. layout improvements
  - made skill search case-insensitive
- *gui/liquids*: added more keybindings: 0-7 to change liquid level, P/B to cycle backwards
- *gui/pathable*: added tile types to sidebar
- *gui/rename*: added "clear" and "special characters" options
- *launch*: can now ride creatures
- ***modtools/skill-change*:**
  - now updates skill levels appropriately
  - only prints output if `-loud` is passed
- *names*: can now edit names of units
- ***remotefortressreader*:**
  - support for moving adventurers
  - support for vehicles, gem shapes, item volume, art images, item improvements
  - includes item stack sizes
  - some performance improvements

Removed

- *tweak*: `kitchen-keys`: Bug 614 fixed in DF 0.44.04
- *warn-stuck-trees*: Bug 9252 fixed in DF 0.44.01

Internals

- `Gui::getAnyUnit()` supports many more screens/menus

Lua

- Added a new `dfhack.console` API
- API can now wrap functions with 12 or 13 parameters
- Exposed `get_vector()` (from C++) for all types that support `find()`, e.g. `df.unit.get_vector() == df.global.world.units.all`
- Improved `json` I/O error messages
- Stopped a crash when trying to create instances of classes whose vtable addresses are not available

Structures

- Added `buildings_other_id.DISPLAY_CASE`
- Added `job_type.PutItemOnDisplay`
- Added `twbt_render_map` code offset on x64
- Fixed an issue preventing `enabler` from being allocated by DFHack
- Fixed `unit` alignment
- Fixed `viewscreen_titlest.start_savegames` alignment
- Found `renderer` vtable on osx64
- Identified `historical_entity.unknown1b.deities` (deity IDs)
- Located `start_dwarf_count` offset for all builds except 64-bit Linux; *startdwarf* should work now
- **New globals:**
  - `soul_next_id`
  - `version`
  - `min_load_version`
  - `movie_version`
  - `basic_seed`
  - `title`
  - `title_spaced`
  - `ui_building_resize_radius`
- The former `announcements` global is now a field in `d_init`
- The `ui_menu_width` global is now a 2-byte array; the second item is the former `ui_area_map_width` global, which is now removed
- `adventure_movement_optionst`, `adventure_movement_hold_tilest`, `adventure_movement_climbst`: named coordinate fields
- `artifact_record`: fixed layout (changed in 0.44.04)

- `incident`: fixed layout (changed in 0.44.01) - note that many fields have moved
- `mission`: added type
- `unit`: added 3 new vmethods: `getCreatureTile`, `getCorpseTile`, `getGlowTile`
- `viewscreen_assign_display_itemst`: fixed layout on x64 and identified many fields
- `viewscreen_reportliststst`: fixed layout, added `mission_id` vector
- `world.status`: named `missions` vector
- `world` fields formerly beginning with `job_` are now fields of `world.jobs`, e.g. `world.job_list` is now `world.jobs.list`

8.1.19 Older Changelogs

Are kept in a seperate file: History

8.2 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	
Abel	abstern	
acwatkins	acwatkins	
Alexander Gavrilov	angavrilov	ag
Amostubal	Amostubal	
Andrea Cattaneo	acattaneo88	
AndreasPK	AndreasPK	
Angus Mezick	amezick	
Antalia	tamarakorr	
Anuradha Dissanayake	falconne	
arzyu	arzyu	
Atkana	Atkana	
AtomicChicken	AtomicChicken	
Bearskie	Bearskie	
belal	jimhester	
Ben Lubar	BenLubar	
Ben Rosser	TC01	
Benjamin McKenna	britishben	
Benjamin Seiller	bseiller	RedDwarfStepper
billw2012	billw2012	
BrickViking	brickviking	
brndd	brndd	burneddi
Caldfir	caldfir	
Carter Bray	Qartar	
Chris Dombroski	cdombroski	
Chris Parsons	chrismdp	
Clayton Hughes		

Continued on next page

Table 1 – continued from previous page

Name	Github	Other
Clément Vuchener	cvuchener	
daedsidog	daedsidog	
Dan Amlund	danamlund	
Daniel Brooks	db48x	
David	Nilsolm	
David Corbett	dscorbett	
David Seguin	dseguin	
David Timm	dtimm	
Deon		
DoctorVanGogh	DoctorVanGogh	
Donald Ruegsegger	hashaash	
doomchild	doomchild	
DwarvenM	DwarvenM	
ElMendukol	ElMendukol	
enjia2000		
Eric Wald	eswald	
Erik Youngren	Artanis	
Espen Wiborg		
expwnent	expwnent	
Feng		
figment	figment	
Gabe Rau	gaberau	
gchristopher	gchristopher	
George Murray	GitOnUp	
grubsteak	grubsteak	
Guilherme Abraham	GuilhermeAbraham	
Harlan Playford	playfordh	
Hayati Ayguen	hayguen	
Herwig Hochleitner	bendlas	
Ian S	kremlin-	
IndigoFenix		
James Gilles	kazimuth	
James Logsdon	jlogsdon	
Jared Adams		
Jeremy Apthorp	nornagon	
Jim Lisi	stonetoad	
Jimbo Whales	jimbowhales	
jimcarreer	jimcarreer	
jj	jjyg	jj‘‘
Joel Meador	janxious	
John Beisley	huin	
John Shade	gsvslto	
Jonas Ask		
Josh Cooper	cppcooper	coope
kane-t	kane-t	
Kelly Kinkade	ab9rf	
KlonZK	KlonZK	
Kris Parker	kaypy	
Kristjan Moore	kristjanmoore	

Continued on next page

Table 1 – continued from previous page

Name	Github	Other
Kromtec	Kromtec	
Kurik Amudnil		
Lethosor	lethosor	
LordGolias	LordGolias	
Mark Nielson	pseudodragon	
Mason11987	Mason11987	
Matt Regul	mattregul	
Matthew Cline		
Matthew Lindner	mlindner	
Matthew Taylor	ymber	yutna
Max	maxthyme	Max^TM
McArcady	McArcady	
melkor217	melkor217	
Meneth32		
Meph		
Michael Casadevall	NCommander	
Michael Crouch	creidieki	
Michon van Dooren	MaienM	
miffedmap	miffedmap	
Mike Stewart	thewonderidiot	
Mikko Juola	Noeda	Adeon
Milo Christiansen	milochristiansen	
MithrilTuxedo	MithrilTuxedo	
mizipzor	mizipzor	
moversti	moversti	
Myk Taylor	myk002	
napagokc	napagokc	
Neil Little	nmlittle	
Nick Rart	nickrart	comestible
Nicolas Ayala	nicolasayala	
Nik Nyby	nikolas	
Nikolay Amiantov	abbradar	
nocico	nocico	
Omniclasm		
OwnageIsMagic	OwnageIsMagic	
palenerd	dlmarquis	
PassionateAngler	PassionateAngler	
Patrik Lundell	PatrikLundell	
Paul Fenwick	pjf	
PeridexisErrant	PeridexisErrant	
Petr Mrázek	peterix	
Pfhreak	Pfhreak	
Pierre Lulé	plule	
Pierre-David Bélanger	pierredavidbelanger	
potato		
Priit Laes	plaes	
Putnam	Putnam3145	
Quietust	quietust	_Q
Raidau	Raidau	

Continued on next page

Table 1 – continued from previous page

Name	Github	Other
Ralph Bisschops	ralpha	
Ramblurr	Ramblurr	
rampaging-poet		
Raoul van Putten		
Raoul XQ	raoulxq	
reverb		
Rich Rauenzahn	rrauenza	
Rinin	Rinin	
rndmvar	rndmvar	
Robert Heinrich	rh73	
Robert Janetzko	robertjanetzko	
Rocco Moretti	roccomoretti	
RocheLimit		
rofl0r	rofl0r	
root		
Rose	RosaryMala	
Roses	Pheosics	
Ross M	RossM	
rout		
rubybrowncoat	rubybrowncoat	
Rumrusher	rumrusher	
RusAnon	RusAnon	
Ryan Bennitt	ryanbennitt	
Ryan Williams	Bumber64	Bumber
sami		
scamtank	scamtank	
Sebastian Wolfertz	Enkrod	
seishuuu	seishuuu	
Seth Woodworth	sethwoodworth	
simon		
Simon Jackson	sizeak	
stolencatkarma		
Stoyan Gaydarov	sgayda2	
Su	Moth-Tolias	
suokko	suokko	shrieker
sv-esk	sv-esk	
Tachytaenius	wolfboyft	
Tacomagic		
thefriendlyhacker	thefriendlyhacker	
TheHologram	TheHologram	
Theo Kalfas	teolandon	
therahedwig	therahedwig	
ThiagoLira	ThiagoLira	
thurin	thurin	
Tim Siegel	softmoth	
Tim Walberg	twalberg	
Timothy Collett	danaris	
Timur Kelman	TymurGubayev	
Tom Jobbins	TheBloke	

Continued on next page

Table 1 – continued from previous page

Name	Github	Other
Tom Prince		
Tommy R	tommy	
TotallyGatsby	TotallyGatsby	
Travis Hoppe	thoppe	orthographic-pedant
txtsd	txtsd	
U-glouglou\simon		
Valentin Ochs	Cat-Ion	
Vitaly Pronkin	pronvit	mifki
ViTuRaS	ViTuRaS	
Vjek	vjek	
Warmist	warmist	
Wes Malone	wesQ3	
Will Rogers	wjrogers	
ZechyW	ZechyW	
Zhentar	Zhentar	
zilpin	zilpin	
Zishi Wu	zishiwu123	

8.3 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
libexpat	MIT	(c) 1998-2000 Thai Open Source Software Center Ltd and Clark Cooper (c) 2001-2019 Expat maintainers
libzip	BSD 3-clause	(c) 1999-2020 Dieter Baron and Thomas Klausner
linenoise	BSD 2-clause	(c) 2010, Salvatore Sanfilippo & Pieter Noordhuis
lua	MIT	(c) 1994-2008, Lua.org, PUC-Rio.
luacov	MIT	(c) 2007 - 2018 Hisham Muhammad
luafilesystem	MIT	(c) 2003-2014, Kepler Project
lua-profiler	MIT	(c) 2002,2003,2004 Pepperfish
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
xlsxio	MIT	(c) 2016-2020, Brecht Sanders
alt-getopt	MIT	(c) 2009 Aleksey Cheusov

8.3.1 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.
```

8.3.2 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.
```

8.3.3 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
```

8.4 Removed tools

This page lists tools (plugins or scripts) that were previously included in DFHack but have been removed. It exists primarily so that internal links still work (e.g. links from the *Changelog*).

Contents

8.4.1 digfort

A script to designate an area for digging according to a plan in csv format. Please use DFHack's more powerful *quickfort* script instead. You can use your existing .csv files. Just move them to the `blueprints` folder in your DF installation, and instead of `digfort file.csv`, run `quickfort run file.csv`.

8.4.2 fortplan

Designates furniture for building according to a `.csv` file with quickfort-style syntax. Please use DFHack's more powerful *quickfort* script instead. You can use your existing .csv files. Just move them to the `blueprints` folder in your DF installation, and instead of `fortplan file.csv` run `quickfort run file.csv`.

8.4.3 warn-stuck-trees

The corresponding DF bug, Bug 9252 was fixed in DF 0.44.01.

CHAPTER 9

DFHack Development Guide

These are pages relevant to people developing for DFHack.

9.1 DFHack development overview

DFHack has various components; this page provides an overview of some. If you are looking to develop a tool for DFHack, developing a script or plugin is likely the most straightforward choice.

Other pages that may be relevant include:

- *How to contribute to DFHack*
- *DFHack Documentation System*
- *Licenses*

Contents

9.1.1 Plugins

DFHack plugins are written in C++ and located in the `plugins` folder. Currently, documentation on how to write plugins is somewhat sparse. There are templates that you can use to get started in the `plugins/skeleton` folder, and the source code of existing plugins can also be helpful.

If you want to compile a plugin that you have just added, you will need to add a call to `DFHACK_PLUGIN` in `plugins/CMakeLists.txt`.

Plugins have the ability to make one or more commands available to users of the DFHack console. Examples include *3dveins* (which implements the `3dveins` command) and *reveal* (which implements `reveal`, `unreveal`, and several other commands).

Plugins can also register handlers to run on every tick, and can interface with the built-in *enable* and *enable* commands. For the full plugin API, see the skeleton plugins or `PluginManager.cpp`.

Installed plugins live in the `hack/plugins` folder of a DFHack installation, and the *load* family of commands can be used to load a recompiled plugin without restarting DF.

See *DFHack Plugins* for a list of all plugins included in DFHack.

9.1.2 Scripts

DFHack scripts can currently be written in Lua or Ruby. The *Lua API* is more complete and currently better-documented, however. Referring to existing scripts as well as the API documentation can be helpful when developing new scripts.

Scripts included in DFHack live in a separate scripts repository. This can be found in the `scripts` submodule if you have *cloned DFHack*, or the `hack/scripts` folder of an installed copy of DFHack.

9.1.3 Core

The *DFHack core* has a variety of low-level functions. It is responsible for hooking into DF (via SDL), providing a console, and providing an interface for plugins and scripts to interact with DF.

9.1.4 Modules

A lot of shared code to interact with DF in more complicated ways is contained in **modules**. For example, the Units module contains functions for checking various traits of units, changing nicknames properly, and more. Generally, code that is useful to multiple plugins and scripts should go in the appropriate module, if there is one.

Several modules are also *exposed to Lua*, although some functions (and some entire modules) are currently only available in C++.

9.1.5 Remote access interface

DFHack provides a remote access interface that external tools can connect to and use to interact with DF. See *DFHack Remote Interface* for more information.

9.2 How to contribute to DFHack

Contents

9.2.1 Contributing Code

DFHack's source code is hosted on GitHub. To obtain the code, you do not need an account - see the *compilation instructions* for details. However, to contribute code to DFHack, you will need a GitHub account to submit pull requests. DFHack consists of several repositories, so you will need to fork the repository (or repositories) containing the code you wish to modify. GitHub has several documentation pages on these topics, including:

- An overview of forks
- Proposing changes with pull requests (note: see *Pull request guidelines* for some DFHack-specific information)

In general, if you are not sure where or how to make a change, or would like advice before attempting to make a change, please see *Getting Support* for ways to contact maintainers - DFHack-specific channels such as IRC or Bay12 are preferred. If you are interested in addressing an issue reported on the issue tracker, you can start a discussion there if you prefer.

The sections below cover some guidelines that contributions should follow:

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
  - `#include` directives should be sorted: C++ libraries first, then DFHack modules, then `df/` headers, then local includes. Within each category they should be sorted alphabetically.

Pull request guidelines

- Pull requests should be based on (and submitted to) the default branch of the relevant repo, which is the branch you see when you access the repo on GitHub or clone the repo without specifying a branch. As of 0.47.04-r1, this is `develop` for the main DFHack repo and `master` for other repos.
- We often leave feedback as comments on pull requests, so be sure that you have notifications turned on or that you check back for feedback periodically.

- 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).
  - An exception: for a collection of small miscellaneous changes (e.g. structures research), one branch instead of many small branches is fine. It is still preferred that this branch be dedicated to this purpose, i.e. not `master` or `develop`. Your pull request may be merged at any point unless you indicate that it isn't ready (see below), but you can continue to push to the same branch and open new pull requests as needed.
- Try to keep pull requests relatively small so that they are easier to review and merge.
  - If you expect to make a large number of related additions or changes (e.g. adding a large new plugin), multiple PRs are preferred, as they allow more frequent (and easier) feedback. If development of this feature is expected to take a while, we may create a dedicated branch to merge your pull requests into instead of the repo's default branch.
- If you plan to make additional changes to your pull request in the near future, or if it isn't quite ready to be merged, mark it as a draft pull request or add "WIP" to the title. Otherwise, your pull request may be reviewed and/or merged prematurely.

General contribution guidelines

- If convenient, compile on multiple platforms when changing anything that compiles. Our CI should catch anything that fails to build, but checking in advance can sometimes let you know of any issues sooner.
- Update documentation when applicable - see *Documentation standards* for details.
- Update `changelog.txt` and `docs/Authors.rst` when applicable. See *Building the changelogs* for more information on the changelog format.
- 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.

9.2.2 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!

9.3 Compiling DFHack

DFHack builds are available for all supported platforms; see *Installing DFHack* for installation instructions. If you are a DFHack end-user, modder, or plan on writing scripts (not plugins), it is generally recommended (and easier) to use these builds instead of compiling DFHack from source.

However, if you are looking to develop plugins, work on the DFHack core, make complex changes to DF-structures, or anything else that requires compiling DFHack from source, this document will walk you through the build process. Note that some steps may be unconventional compared to other projects, so be sure to pay close attention if this is your first time compiling DFHack.

Contents

9.3.1 How to get the code

DFHack uses Git for source control; instructions for installing Git can be found in the platform-specific sections below. The code is hosted on GitHub, and can be downloaded with:

```
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.

This will check out the code on the default branch of the GitHub repo, currently `develop`, which may be unstable. If you want code for the latest stable release, you can check out the `master` branch instead:

```
git checkout master
git submodule update
```

In general, a single DFHack clone is suitable for development - most Git operations such as switching branches can be done on an existing clone. If you find yourself cloning DFHack frequently as part of your development process, or getting stuck on anything else Git-related, feel free to reach out to us for assistance.

Offline builds

If you plan to build DFHack on a machine without an internet connection (or with an unreliable connection), see *Note on building DFHack offline* for additional instructions.

Working with submodules

DFHack uses submodules extensively to manage its subprojects (including the `scripts` folder and DF-structures in `library/xml`). Failing to keep submodules in sync when switching between branches can result in build errors or scripts that don't work. In general, you should always update submodules whenever you switch between branches in the main DFHack repo with `git submodule update`. (If you are working on bleeding-edge DFHack and have checked out the master branch of some submodules, running `git pull` in those submodules is also an option.)

Rarely, we add or remove submodules. If there are any changes to the existence of submodules when you switch between branches, you should run `git submodule update --init` instead (adding `--init` to the above command).

Some common errors that can arise when failing to update submodules include:

- `fatal: <some path> does not exist` when performing Git operations
- Build errors, particularly referring to structures in the `df::` namespace or the `library/include/df` folder
- `Not a known DF version` when starting DF
- `Run 'git submodule update --init'` when running CMake

Submodules are a particularly confusing feature of Git. The Git Book has a thorough explanation of them (as well as of many other aspects of Git) and is a recommended resource if you run into any issues. Other DFHack developers are also able to help with any submodule-related (or Git-related) issues you may encounter.

Contributing to DFHack

For details on contributing to DFHack, including pull requests, code format, and more, please see *Contributing Code*.

9.3.2 Build settings

This section describes build configuration options that apply to all platforms. If you don't have a working build environment set up yet, follow the instructions in the platform-specific sections below first, then come back here.

Generator

The `Ninja` CMake build generator is the preferred build method on Linux and macOS, instead of `Unix Makefiles`, which is the default. You can select Ninja by passing `-G Ninja` to CMake. Incremental builds using Unix Makefiles can be much slower than Ninja builds. Note that you will probably need to install Ninja; see the platform-specific sections for details.

```
cmake .. -G Ninja
```

Warning: Most other CMake settings can be changed by running `cmake` again, but the generator cannot be changed after `cmake` has been run without creating a new build folder. Do not forget to specify this option.

CMake versions 3.6 and older, and possibly as recent as 3.9, are known to produce project files with dependency cycles that fail to build (see Issue 1369). Obtaining a recent version of CMake is recommended, either from cmake.org or through a package manager. See the sections below for more platform-specific directions for installing CMake.

Build type

`cmake` allows you to pick a build type by changing the `CMAKE_BUILD_TYPE` variable:

```
cmake .. -DCMAKE_BUILD_TYPE:string=BUILD_TYPE
```

Valid and useful build types include 'Release' and 'RelWithDebInfo'. The default build type is 'Release'.

Target architecture (32-bit vs. 64-bit)

Set DFHACK_BUILD_ARCH to either `32` or `64` to build a 32-bit or 64-bit version of DFHack (respectively). The default is currently `64`, so you will need to specify this explicitly for 32-bit builds. Specifying it is a good idea in any case.

```
cmake .. -DDFHACK_BUILD_ARCH=32
```

or

```
cmake .. -DDFHACK_BUILD_ARCH=64
```

Note that the scripts in the "build" folder on Windows will set the architecture automatically.

Other settings

There are a variety of other settings which you can find in CMakeCache.txt in your build folder or by running `ccmake` (or another CMake GUI). Most DFHack-specific settings begin with `BUILD_` and control which parts of DFHack are built.

9.3.3 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 need to be installed with your package manager.)

To build DFHack, you need GCC 4.8 or newer. GCC 4.8 has the benefit of avoiding *libstdc++ compatibility issues*, but can be hard to obtain on modern distributions, and working around these issues is done automatically by the `dfhack` launcher script. As long as your system-provided GCC is new enough, it should work. Note that extremely new GCC versions may not have been used to build DFHack yet, so if you run into issues with these, please let us know (e.g. by opening a GitHub issue).

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. As mentioned above, `ninja` is recommended (many distributions call this package `ninja-build`).

You will need pthread; most systems should have this already. Note that older CMake versions may have trouble detecting pthread, so if you run into pthread-related errors and pthread is installed, you may need to upgrade CMake, either by downloading it from cmake.org or through your package manager, if possible.

You also need zlib, libsdl (1.2, not sdl2, like DF), perl, and the XML::LibXML and XML::LibXSLT perl packages (for the code generation parts). You should be able to find them in your distribution's repositories.

To build stonesense, you'll also need OpenGL headers.

Here are some package install commands for various distributions:

- On Arch linux:
  - For the required Perl modules: `perl-xml-libxml` and `perl-xml-libxslt` (or through `cpan`)
- On Ubuntu:

```
apt-get install gcc cmake ninja-build git zlib1g-dev libsdl1.2-dev libxml-libxml-
↪perl libxml-libxslt-perl
```

  - Other Debian-based distributions should have similar requirements.

- On Fedora:

```
yum install gcc-c++ cmake ninja-build git zlib-devel SDL-devel perl-core perl-XML-
↪LibXML perl-XML-LibXSLT ruby
```

Multilib dependencies

If you want to compile 32-bit DFHack 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 4.8 on a distribution that defaults to 5.x - you may need to add the version number to the multilib packages.
  - For example, `gcc-4.8-multilib` and `g++-4.8-multilib` if installing for GCC 4.8 on a system that uses a later GCC version.
  - This is definitely required on Ubuntu/Debian, check if using a different distribution.
- `zlib1g-dev:i386` (or a similar i386 zlib-dev package)

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.

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 .. -G Ninja -DCMAKE_BUILD_TYPE:string=Release -DCMAKE_INSTALL_PREFIX=<path to␣
↪DF>
ninja install  # or ninja -jX install to specify the number of cores (X) to use
```

<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 .. -G Ninja
ninja 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 in the `libs` folder, which comes from GCC 4.8 and is incompatible with code compiled with newer GCC versions. As of DFHack 0.42.05-alpha1, the `dfhack` launcher script attempts to fix this by automatically removing the DF-provided libstdc++ on startup. In rare cases, this may fail and cause errors such as:

```
./libs/Dwarf_Fortress: /pathToDF/libs/libstdc++.so.6: version
    `GLIBCXX_3.4.18' not found (required by ./hack/libdfhack.so)
```

The easiest way to fix this is generally removing 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 may result in the opposite compatibily issue: users with *older* GCC versions may encounter similar errors. This is why DFHack distributes both GCC 4.8 and GCC 7 builds. If you are planning on distributing binaries to other users, we recommend using an older GCC (but still at least 4.8) version if possible.

9.3.4 macOS

DFHack functions similarly on macOS and Linux, and the majority of the information above regarding the build process (CMake and Ninja) applies here as well.

DFHack can officially be built on macOS only with GCC 4.8 or 7. Anything newer than 7 will require you to perform extra steps to get DFHack to run (see *Notes for GCC 8+ or OS X 10.10+ users*), and your build will likely not be redistributable.

Notes for GCC 8+ or OS X 10.10+ users

If none of these situations apply to you, skip to *Dependencies and system set-up*.

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
```

If you build with a GCC version newer than 7, DFHack will probably crash immediately on startup, or soon after. To fix this, you will need to replace `hack/libstdc++.6.dylib` with a symlink to the `libstdc++.6.dylib` included in your version of GCC:

```
cd <path to df>/hack && mv libstdc++.6.dylib libstdc++.6.dylib.orig &&
ln -s [PATH_TO_LIBSTDC++] .
```

For example, with GCC 6.3.0, `PATH_TO_LIBSTDC++` would be:

```
/usr/local/Cellar/gcc@6/6.3.0/lib/gcc/6/libstdc++.6.dylib  # for 64-bit DFHack
/usr/local/Cellar/gcc@6/6.3.0/lib/gcc/6/i386/libstdc++.6.dylib  # for 32-bit DFHack
```

Note: If you build with a version of GCC that requires this, your DFHack build will *not* be redistributable. (Even if you copy the `libstdc++.6.dylib` from your GCC version and distribute that too, it will fail on older OS X versions.) For this reason, if you plan on distributing DFHack, it is highly recommended to use GCC 4.8 or 7.

Notes for M1 users

Alongside the above, you will need to follow these additional steps to get it running on Apple silicon.

Install an x86 copy of `homebrew` alongside your existing one. This stackoverflow answer describes the process.

Follow the normal macOS steps to install `cmake` and `gcc` via your x86 copy of `homebrew`. Note that this will install a GCC version newer than 7, so see *Notes for GCC 8+ or OS X 10.10+ users*.

In your terminal, ensure you have your path set to the correct homebrew in addition to the normal `CC` and `CXX` flags above:

```
export PATH=/usr/local/bin:$PATH
```

Dependencies and system set-up

1. Download and unpack a copy of the latest DF
2. Install Xcode from the Mac App Store
3. Install the XCode Command Line Tools by running the following command:

   ```
   xcode-select --install
   ```

4. Install dependencies

   It is recommended to use Homebrew instead of MacPorts, as it is generally cleaner, quicker, and smarter. For example, installing MacPort's GCC will install more than twice as many dependencies as Homebrew's will, and all in both 32-bit and 64-bit variants. Homebrew also doesn't require constant use of `sudo`.

   Using Homebrew (recommended):

   ```
   brew tap homebrew/versions
   brew install git
   brew install cmake
   brew install ninja
   brew install gcc@7
   ```

   Using MacPorts:

   ```
   sudo port install gcc7 +universal cmake +universal git-core +universal␣
   ↪ninja +universal
   ```

   Macports will take some time - maybe hours. At some point it may ask you to install a Java environment; let it do so.

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 https://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-7
export CXX=/usr/local/bin/g++-7
```

  Macports:

```
export CC=/opt/local/bin/gcc-mp-7
export CXX=/opt/local/bin/g++-mp-7
```

  Change the version numbers appropriately if you installed a different version of GCC.

  If you are confident that you have GCC in your path, you can omit the absolute paths:

```
export CC=gcc-7
export CXX=g++-7
```

  (adjust as needed for different GCC installations)

- Build DFHack:

```
mkdir build-osx
cd build-osx
cmake .. -G Ninja -DCMAKE_BUILD_TYPE:string=Release -DCMAKE_INSTALL_PREFIX=<path␣
↪to DF>
ninja install  # or ninja -jX install to specify the number of cores (X) to use
```

  <path to DF> should be a path to a copy of Dwarf Fortress, of the appropriate version for the DFHack you are building.

9.3.5 Windows

On Windows, DFHack replaces the SDL library distributed with DF.

Dependencies

You will need the following:

- Microsoft Visual C++ 2022, 2019, 2017, or 2015 (optional)
- Microsoft Visual C++ 2015 Build Tools
- Git
- CMake
- Perl with XML::LibXML and XML::LibXSLT
  - It is recommended to install StrawberryPerl, which includes both.
- Python (for documentation; optional, except for release builds)

Microsoft Visual Studio

Releases of Dwarf Fortress since roughly 2016 have been compiled for Windows using Microsoft's Visual Studio 2015 C++ compiler. In order to guarantee ABI and STL compatibility with Dwarf Fortress, DFHack has to be compiled with the same compiler.

Visual Studio 2015 is no longer supported by Microsoft and it can be difficult to obtain working installers for this product today. As of 2022, the recommended approach is to use Visual Studio 2022 or Visual Studio 2019, installing additional optional Visual Studio components which provide the required support for using Visual Studio 2015's toolchain. All of the required tools are available from Microsoft as part of Visual Studio's Community Edition at no charge.

You can also download just the Visual C++ 2015 build tools if you aren't going to use Visual Studio to edit code.

Option 1: Build Tools Only

Click build tools and you will be prompted to login to your Microsoft account. Then you should be redirected to a page with various download options with 2015 in their name. If this redirect doesn't occur, just copy, paste, and enter the download link again and you should see the options. You need to get: Visual C++ Build Tools for Visual Studio 2015 with Update 3. Click the download button next to it and a dropdown of download formats will appear. Select the DVD format to download an ISO file. When the donwload is complete, click on the ISO file and a folder will popup with the following contents:

- packages (folder)
- VCPlusPlusBuildTools2015Update3_x64_Files.cat
- VisualCppBuildTools_Full.exe

The packages folder contains the dependencies that are required by the build tools. These include:

- Microsoft .NET Framework 4.6.1 Developer Pack
- Microsoft Visual C++ 2015 Redistributable (x64) - 14.0.24210
- Windows 10 Universal SDK - 10.0.10240
- Windows 8.1 SDK

Click VisualCppBuildTools_Full.exe and use the default options provided by the installer wizard that appears. After the installation is completed, add the path where MSBuild.exe was installed to your PATH environment variable. The path should be:

- `C:\Program Files (x86)\MSBuild\14.0\Bin`

Note that this process may install only the `v140` toolchain, not the `v140_xp` toolchain that is normally used to compile build releases of DFHack. Due to a bug in the Microsoft-provided libraries used with the `v140_xp` toolchain that Microsoft has never fixed, DFHack (and probably also Dwarf Fortress itself) doesn't run reliably on 64-bit XP. Investigations have so far suggested that `v140` and `v140_xp` are ABI-compatible. As such, there should be no harm in using `v140` instead of `v140_xp` as the build toolchain, at least on 64-bit platforms. However, it is our policy to use `v140_xp` for release builds for both 32-bit and 64-bit Windows, since 32-bit releases of Dwarf Fortress work on XP and `v140_xp` is required for compatibility with XP.

The `v141` toolchain, in Visual Studio 2017, has been empirically documented to be incompatible with released versions of Dwarf Fortress and cannot be used to make usable builds of DFHack.

Option 2: IDE + Build Tools

Click Visual Studio 2022 or 2019 to download an installer wizard that will prompt you to select the optional tools you want to download alongside the IDE. You may need to log into (or create) a Microsoft account in order to download Visual Studio.

In addition to selecting the workload for "Desktop Development with C++", you will also need to go to the "Individual Components" tab in the Installer and select the following additional components to get the "`v140_xp`" toolchain that DFHack requires for ABI compatibility with recent releases of Dwarf Fortress: * MSVC v140 - VS 2015 C++ build tools (v14.00) * C++ Windows XP Support for VS 2017 (v141) tools [Deprecated]

Yes, this is unintuitive. Installing XP Support for VS 2017 installs XP Support for VS 2015 if the 2015 toolchain is installed.

Additional dependencies: installing with the Chocolatey Package Manager

The remainder of dependencies - Git, CMake, StrawberryPerl, and Python - 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 recommended way of installing the required dependencies as it's quicker, requires 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 stage of the build process, 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 directories must be in your PATH, in this order:

- `<path to perl>\c\bin`
- `<path to perl>\perl\site\bin`
- `<path to perl>\perl\bin`
- `<path to perl>\perl\vendor\lib\auto\XML\LibXML` (may only be required on some systems)

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 installed (for example, 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 `win32` and `win64` subfolders in the `build` folder, along with a script that's used for picking the DF path. Use the subfolder corresponding to the architecture that you want to build for.

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.
- `release` will create a solution with everything that should be included in release builds of DFHack. Note that this includes documentation, which requires Python.

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 VC2015 or VC2015_32, depending on the architecture you specified. Open the file `dfhack.sln` inside that folder. If you have multiple versions of Visual Studio installed, make sure you open with Visual Studio 2015.

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`.

9.3.6 Building the documentation

The steps above will not build DFHack's documentation by default. If you are editing documentation, see *DFHack Documentation System* for details on how to build it.

9.3.7 Misc. Notes

Note on building DFHack offline

As of 0.43.05, DFHack downloads several files during the build process, depending on your target OS and architecture. If your build machine's internet connection is unreliable, or nonexistent, you can download these files in advance.

First, you must locate the files you will need. These can be found in the dfhack-bin repo. Look for the most recent version number *before or equal to* the DF version which you are building for. For example, suppose "0.43.05" and "0.43.07" are listed. You should choose "0.43.05" if you are building for 0.43.05 or 0.43.06, and "0.43.07" if you are building for 0.43.07 or 0.43.08.

Then, download all of the files you need, and save them to `<path to DFHack clone>/CMake/downloads/<any filename>`. The destination filename you choose does not matter, as long as the files end up in the `CMake/downloads` folder. You need to download all of the files for the architecture(s) you are building for. For example, if you are building for 32-bit Linux and 64-bit Windows, download all files starting with `linux32` and `win64`. GitHub should sort files alphabetically, so all the files you need should be next to each other.

Note:

- Any files containing "allegro" in their filename are only necessary for building stonesense. If you are not building Stonesense, you don't have to download these, as they are larger than any other listed files.

It is recommended that you create a build folder and run CMake to verify that you have downloaded everything at this point, assuming your download machine has CMake installed. This involves running a "generate" batch script on Windows, or a command starting with `cmake .. -G Ninja` on Linux and macOS, following the instructions in the sections above. CMake should automatically locate files that you placed in `CMake/downloads`, and use them instead of attempting to download them.

9.4 Development Changelog

This file contains changes grouped by the release (stable or development) in which they first appeared. See *Building the changelogs* for more information.

See *Changelog* for a list of changes grouped by stable releases.

Contents

- *DFHack 0.44.02-alpha1*

9.4.1 DFHack 0.47.05-r5

New Plugins

- *spectate*: "spectator mode" – automatically follows dwarves doing things in your fort

New Scripts

- *devel/eventful-client*: useful for testing eventful events

New Tweaks

- *tweak*: `partial-items` displays percentage remaining for partially-consumed items such as hospital cloth

Fixes

- *autofarm*: removed restriction on only planting "discovered" plants
- *cxxrandom*: fixed exception when calling `bool_distribution`
- ***devel/query***:
  - fixed a problem printing parents when the starting path had lua pattern special characters in it
  - fixed a crash when trying to iterate over linked lists
- *gui/advfort*: encrust and stud jobs no longer consume reagents without actually improving the target item
- *luasocket*: return correct status code when closing socket connections so clients can know when to retry
- *quickfort*: contructions and bridges are now properly placed over natural ramps
- *setfps*: keep internal ratio of processing FPS to graphics FPS in sync when updating FPS

Misc Improvements

- ***autochop***:
  - only designate the amount of trees required to reach `max_logs`
  - preferably designate larger trees over smaller ones
- ***autonick***:
  - now displays help instead of modifying dwarf nicknames when run without parameters. use `autonick all` to rename all dwarves.
  - added `--quiet` and `--help` options
- ***blueprint***:
  - `track` phase renamed to `carve`
  - carved fortifications and (optionally) engravings are now captured in generated blueprints

- *cursecheck*: new option, `--ids` prints creature and race IDs of the cursed creature
- ***debug*:**
  - DFHack log messages now have configurable headers (e.g. timestamp, origin plugin name, etc.) via the `debugfilter` command of the *debug* plugin
  - script execution log messages (e.g. "Loading script: dfhack_extras.init" can now be controlled with the `debugfilter` command. To hide the messages, add this line to your `dfhack.init` file: `debugfilter set Warning core script`
- ***DFHack Example Configuration File Index*:**
  - add mugs to `basic` manager orders
  - `onMapLoad_dreamfort.init` remove "cheaty" commands and new tweaks that are now in the default `dfhack.init-example` file
- *dig-now*: handle fortification carving
- ***Events from EventManager*:**
  - add new event type `JOB_STARTED`, triggered when a job first gains a worker
  - add new event type `UNIT_NEW_ACTIVE`, triggered when a new unit appears on the active list
- *gui/blueprint*: support new *blueprint* options and phases
- *gui/create-item*: Added "(chain)" annotation text for armours with the [CHAIN_METAL_TEXT] flag set
- *manipulator*: tweak colors to make the cursor easier to locate
- ***quickfort*:**
  - support transformations for blueprints that use expansion syntax
  - adjust direction affinity when transforming buildings (e.g. bridges that open to the north now open to the south when rotated 180 degrees)
  - automatically adjust cursor movements on the map screen in `#query` and `#config` modes when the blueprint is transformed. e.g. `{Up}` will be played back as `{Right}` when the blueprint is rotated clockwise and the direction key would move the map cursor
  - new blueprint mode: `#config`; for playing back key sequences that don't involve the map cursor (like configuring hotkeys, changing standing orders, or modifying military uniforms)
  - API function `apply_blueprint` can now take `data` parameters that are simple strings instead of coordinate maps. This allows easier application of blueprints that are just one cell.
- *stocks*: allow search terms to match the full item label, even when the label is truncated for length
- *tweak*: `stable-cursor` now keeps the cursor stable even when the viewport moves a small amount
- `dfhack.init-example`: recently-added tweaks added to example `dfhack.init` file

API

- add functions reverse-engineered from ambushing unit code: `Units::isHidden()`, `Units::isFortControlled()`, `Units::getOuterContainerRef()`, `Items::getOuterContainerRef()`

Lua

- *custom-raw-tokens*: library for accessing tokens added to raws by mods
- `dfhack.units`: Lua wrappers for functions reverse-engineered from ambushing unit code: `isHidden(unit)`, `isFortControlled(unit)`, `getOuterContainerRef(unit)`, `getOuterContainerRef(item)`
- `dialogs`: `show*` functions now return a reference to the created dialog
- `dwarfmode.enterSidebarMode()`: passing `df.ui_sidebar_mode.DesignateMine` now always results in you entering `DesignateMine` mode and not `DesignateChopTrees`, even when you looking at the surface (where the default designation mode is `DesignateChopTrees`)
- **`dwarfmode.MenuOverlay`:**
  - if `sidebar_mode` attribute is set, automatically manage entering a specific sidebar mode on show and restoring the previous sidebar mode on dismiss
  - new class function `renderMapOverlay` to assist with painting tiles over the visible map
- `ensure_key`: new global function for retrieving or dynamically creating Lua table mappings
- `safe_index`: now properly handles lua sparse tables that are indexed by numbers
- `string`: new function `escape_pattern()` escapes regex special characters within a string
- **`widgets`:**
  - unset values in `frame_inset` table default to `0`
  - `FilteredList` class now allows all punctuation to be typed into the filter and can match search keys that start with punctuation
  - minimum height of `ListBox` dialog is now calculated correctly when there are no items in the list (e.g. when a filter doesn't match anything)
  - if `autoarrange_subviews` is set, `Panels` will now automatically lay out widgets vertically according to their current height. This allows you to have widgets dynamically change height or become visible/hidden and you don't have to worry about recalculating frame layouts
  - new class `ResizingPanel` (subclass of `Panel`) automatically recalculates its own frame height based on the size, position, and visibility of its subviews
  - new class `HotkeyLabel` (subclass of `Label`) that displays and reacts to hotkeys
  - new class `CycleHotkeyLabel` (subclass of `Label`) allows users to cycle through a list of options by pressing a hotkey
  - new class `ToggleHotkeyLabel` (subclass of `CycleHotkeyLabel`) toggles between `On` and `Off` states
  - new class `WrappedLabel` (subclass of `Label`) provides autowrapping of text
  - new class `TooltipLabel` (subclass of `WrappedLabel`) provides tooltip-like behavior

Structures

- `adventure_optionst`: add missing `getUnitContainer` vmethod
- `historical_figure.T_skills`: add `account_balance` field
- `job`: add `improvement` field (union with `hist_figure_id` and `race`)
- `report_init.flags`: rename `sparring` flag to `hostile_combat`

- `viewscreen_loadgamest`: add missing `LoadingImageSets` and `LoadingDivinationSets` enum values to `cur_step` field

Documentation

- add more examples to the plugin example skeleton files so they are more informative for a newbie
- update download link and installation instructions for Visual C++ 2015 build tools on Windows
- update information regarding obtaining a compatible Windows build environment
- *confirm*: correct the command name in the plugin help text
- *cxxrandom*: added usage examples
- *String class extentions*: document DFHack string extensions (`startswith()`, `endswith()`, `split()`, `trim()`, `wrap()`, and `escape_pattern()`)
- *Quickfort Blueprint Editing Guide*: added screenshots to the Dreamfort case study and overall clarified text
- *Client libraries*: add new Rust client library
- `Lua API.rst`: added `isHidden(unit)`, `isFortControlled(unit)`, `getOuterContainerRef(unit)`, `getOuterContainerRef(item)`

9.4.2 DFHack 0.47.05-r4

Fixes

- ***blueprint*:**
  - fixed passing incorrect parameters to *gui/blueprint* when you run `blueprint gui` with optional params
  - key sequences for constructed walls and down stairs are now correct
- *exportlegends*: fix issue where birth year was outputted as birth seconds
- ***quickfort*:**
  - produce a useful error message instead of a code error when a bad query blueprint key sequence leaves the game in a mode that does not have an active cursor
  - restore functionality to the `--verbose` commandline flag
  - don't designate tiles for digging if they are within the bounds of a planned or constructed building
  - allow grates, bars, and hatches to be built on flat floor (like DF itself allows)
  - allow tracks to be built on hard, natural rock ramps
  - allow dig priority to be properly set for track designations
  - fix incorrect directions for tracks that extend south or east from a track segment pair specified with expansion syntax (e.g. T(4x4))
  - fix parsing of multi-part extended zone configs (e.g. when you set custom supply limits for hospital zones AND set custom flags for a pond)
  - fix error when attempting to set a custom limit for plaster powder in a hospital zone
- *tailor*: fixed some inconsistencies (and possible crashes) when parsing certain subcommands, e.g. `tailor help`

- *tiletypes-here*, *tiletypes-here-point*: fix crash when running from an unsuspended core context

Misc Improvements

- Core: DFHack now prints the name of the init script it is running to the console and stderr
- *automaterial*: ensure construction tiles are laid down in order when using *buildingplan* to plan the constructions
- ***blueprint***:
  - all blueprint phases are now written to a single file, using *quickfort* multi-blueprint file syntax. to get the old behavior of each phase in its own file, pass the `--splitby=phase` parameter to `blueprint`
  - you can now specify the position where the cursor should be when the blueprint is played back with *quickfort* by passing the `--playback-start` parameter
  - generated blueprints now have labels so *quickfort* can address them by name
  - all building types are now supported
  - multi-type stockpiles are now supported
  - non-rectangular stockpiles and buildings are now supported
  - blueprints are no longer generated for phases that have nothing to do (unless those phases are explicitly enabled on the commandline or gui)
  - new "track" phase that discovers and records carved tracks
  - new "zone" phase that discovers and records activity zones, including custom configuration for ponds, gathering, and hospitals
- *dig-now*: no longer leaves behind a designated tile when a tile was designated beneath a tile designated for channeling
- ***gui/blueprint***:
  - support the new `--splitby` and `--format` options for *blueprint*
  - hide help text when the screen is too short to display it
- *orders*: added `list` subcommand to show existing exported orders
- *Blueprint Library Index*: added light aquifer tap and pump stack blueprints (with step-by-step usage guides) to the quickfort blueprint library
- ***quickfort***:
  - Dreamfort blueprint set improvements: added iron and flux stock level indicators on the industry level and a prisoner processing quantum stockpile in the surface barracks. also added help text for how to manage sieges and how to manage prisoners after a siege.
  - add `quickfort.apply_blueprint()` API function that can be called directly by other scripts
  - by default, don't designate tiles for digging that have masterwork engravings on them. quality level to preserve is configurable with the new `--preserve-engravings` param
  - implement single-tile track aliases so engraved tracks can be specified tile-by-tile just like constructed tracks
  - allow blueprints to jump up or down multiple z-levels with a single command (e.g. `#>5` goes down 5 levels)

  - blueprints can now be repeated up and down a specified number of z-levels via `repeat` markers in meta blueprints or the `--repeat` commandline option
  - blueprints can now be rotated, flipped, and shifted via `transform` and `shift` markers in meta blueprints or the corresponding commandline options
- *quickfort*, *DFHack Example Configuration File Index*: Dreamfort blueprint set improvements based on playtesting and feedback. includes updated profession definitions.

Removed

- *digfort*: please use *quickfort* instead
- *fortplan*: please use *quickfort* instead

API

- `Buildings::findCivzonesAt()`: lookups now complete in constant time instead of linearly scanning through all civzones in the game
- `Job::remove_postings()`: use the job cancel vmethod graciously provided by The Toady One in place of a synthetic method derived from reverse engineering

Lua

- `argparse.processArgsGetopt()`: you can now have long form parameters that are not an alias for a short form parameter. For example, you can now have a parameter like `--longparam` without needing to have an equivalent one-letter `-l` param.
- `dwarfmode.enterSidebarMode()`: `df.ui_sidebar_mode.DesignateMine` is now a suported target sidebar mode

Structures

- `historical_figure_info.spheres`: give spheres vector a usable name
- `unit.enemy`: fix definition of `enemy_status_slot` and add `combat_side_id`

9.4.3 DFHack 0.47.05-r3

New Plugins

- *dig-now*: instantly completes dig designations (including smoothing and carving tracks)

New Scripts

- *autonick*: gives dwarves unique nicknames
- *build-now*: instantly completes planned building constructions
- *do-job-now*: makes a job involving current selection high priority
- *prioritize*: automatically boosts the priority of current and/or future jobs of specified types, such as hauling food, tanning hides, or pulling levers

- *reveal-adv-map*: exposes/hides all world map tiles in adventure mode

Fixes

- Core: `alt` keydown state is now cleared when DF loses and regains focus, ensuring the `alt` modifier state is not stuck on for systems that don't send standard keyup events in response to `alt-tab` window manager events
- Lua: `memscan.field_offset()`: fixed an issue causing *devel/export-dt-ini* to crash sometimes, especially on Windows
- *autofarm*: autofarm will now count plant growths as well as plants toward its thresholds
- *autogems*: no longer assigns gem cutting jobs to workshops with gem cutting prohibited in the workshop profile
- *devel/export-dt-ini*: fixed incorrect vtable address on Windows
- ***quickfort***:
  - allow machines (e.g. screw pumps) to be built on ramps just like DF allows
  - fix error message when the requested label is not found in the blueprint file

Misc Improvements

- *assign-beliefs*, *assign-facets*: now update needs of units that were changed
- *buildingplan*: now displays which items are attached and which items are still missing for planned buildings
- ***devel/query***:
  - updated script to v3.2 (i.e. major rewrite for maintainability/readability)
  - merged options `-query` and `-querykeys` into `-search`
  - merged options `-depth` and `-keydepth` into `-maxdepth`
  - replaced option `-safer` with `-excludetypes` and `-excludekinds`
  - improved how tile data is dealt with identification, iteration, and searching
  - added option `-findvalue`
  - added option `-showpaths` to print full data paths instead of nested fields
  - added option `-nopointers` to disable printing values with memory addresses
  - added option `-alignto` to set the value column's alignment
  - added options `-oneline` and alias `-1` to avoid using two lines for fields with metadata
  - added support for matching multiple patterns
  - added support for selecting the highlighted job, plant, building, and map block data
  - added support for selecting a Lua script (e.g. *dorf_tables*)
  - added support for selecting a Json file (e.g. dwarf_profiles.json)
  - removed options `-listall`, `-listfields`, and `-listkeys` - these are now simply default behaviour
  - `-table` now accepts the same abbreviations (global names, `unit`, `screen`, etc.) as *lua* and *gui/gm-editor*
- *dorf_tables*: integrated *devel/query* to show the table definitions when requested with `-list`

- *geld*: fixed `-help` option
- *gui/gm-editor*: made search case-insensitive
- ***orders*:**
  - support importing and exporting reaction-specific item conditions, like "lye-containing" for soap production orders
  - new `sort` command. sorts orders according to their repeat frequency. this prevents daily orders from blocking other orders for simlar items from ever getting completed.
- ***quickfort*:**
  - Dreamfort blueprint set improvements: extensive revision based on playtesting and feedback. includes updated `onMapLoad_dreamfort.init` settings file, enhanced automation orders, and premade profession definitions. see full changelog at https://github.com/DFHack/dfhack/pull/1921 and https://github.com/DFHack/dfhack/pull/1925
  - accept multiple commands, list numbers, and/or blueprint lables on a single commandline
- *tailor*: allow user to specify which materials to be used, and in what order
- *tiletypes-here*, *tiletypes-here-point*: add `--cursor` and `--quiet` options to support non-interactive use cases
- *unretire-anyone*: replaced the 'undead' descriptor with 'reanimated' to make it more mod-friendly
- *warn-starving*: added an option to only check sane dwarves

API

- The `Items` module `moveTo*` and `remove` functions now handle projectiles

Internals

- Install tests in the scripts repo into hack/scripts/test/scripts when the CMake variable BUILD_TESTS is defined

Lua

- new global function: `safe_pairs(iterable[, iterator_fn])` will iterate over the `iterable` (a table or iterable userdata) with the `iterator_fn` (`pairs` if not otherwise specified) if iteration is possible. If iteration is not possible or would throw an error, for example if `nil` is passed as the `iterable`, the iteration is just silently skipped.

Structures

- `cursed_tomb`: new struct type
- `job_item`: identified several fields
- `ocean_wave_maker`: new struct type
- `worldgen_parms`: moved to new struct type

Documentation

- *DFHack Example Configuration File Index*: documentation for all of *Dreamfort*'s supporting files (useful for all forts, not just Dreamfort!)
- *Blueprint Library Index*: updated dreamfort documentation and added screenshots

9.4.4 DFHack 0.47.05-r2

New Scripts

- *clear-webs*: removes all webs on the map and/or frees any webbed creatures
- *devel/block-borders*: overlay that displays map block borders
- *devel/luacov*: generate code test coverage reports for script development. Define the `DFHACK_ENABLE_LUACOV=1` environment variable to start gathering coverage metrics.
- *fix/drop-webs*: causes floating webs to fall to the ground
- *gui/blueprint*: interactive frontend for the *blueprint* plugin (with mouse support!)
- *gui/mass-remove*: mass removal/suspension tool for buildings and constructions
- *reveal-hidden-sites*: exposes all undiscovered sites
- *set-timeskip-duration*: changes the duration of the "Updating World" process preceding the start of a new game, enabling you to jump in earlier or later than usual

Fixes

- Fixed an issue preventing some external scripts from creating zones and other abstract buildings (see note about room definitions under "Internals")
- Fixed an issue where scrollable text in Lua-based screens could prevent other widgets from scrolling
- ***bodyswap*:**
  - stopped prior party members from tagging along after bodyswapping and reloading the map
  - made companions of bodyswapping targets get added to the adventurer party - they can now be viewed using the in-game party system
- ***buildingplan*:**
  - fixed an issue where planned constructions designated with DF's sizing keys (`umkh`) would sometimes be larger than requested
  - fixed an issue preventing other plugins like *automaterial* from planning constructions if the "enable all" buildingplan setting was turned on
  - made navigation keys work properly in the materials selection screen when alternate keybindings are used
- *color-schemes*: fixed an error in the `register` subcommand when the DF path contains certain punctuation characters
- *command-prompt*: fixed issues where overlays created by running certain commands (e.g. *gui/liquids*, *gui/teleport*) would not update the parent screen correctly
- *dwarfvet*: fixed a crash that could occur with hospitals overlapping with other buildings in certain ways

- *embark-assistant*: fixed faulty early exit in first search attempt when searching for waterfalls
- *gui/advfort*: fixed an issue where starting a workshop job while not standing at the center of the workshop required advancing time manually
- *gui/unit-info-viewer*: fixed size description displaying unrelated values instead of size
- *orders*: fixed crash when importing orders with malformed IDs
- ***quickfort***:
  - comments in blueprint cells no longer prevent the rest of the row from being read. A cell with a single '#' marker in it, though, will still stop the parser from reading further in the row.
  - fixed an off-by-one line number accounting in blueprints with implicit `#dig` modelines
  - changed to properly detect and report an error on sub-alias params with no values instead of just failing to apply the alias later (if you really want an empty value, use `{Empty}` instead)
  - improved handling of non-rectangular and non-solid extent-based structures (like fancy-shaped stockpiles and farm plots)
  - fixed conversion of numbers to DF keycodes in `#query` blueprints
  - fixed various errors with cropping across the map edge
  - properly reset config to default values in `quickfort reset` even if if the `dfhack-config/quickfort/quickfort.txt` config file doesn't mention all config vars. Also now works even if the config file doesn't exist.
- stonesense: fixed a crash that could occur when ctrl+scrolling or closing the Stonesense window
- `quickfortress.csv` blueprint: fixed refuse stockpile config and prevented stockpiles from covering stairways

Misc Improvements

- Added adjectives to item selection dialogs, used in tools like *gui/create-item* - this makes it possible to differentiate between different types of high/low boots, shields, etc. (some of which are procedurally generated)
- ***blueprint***:
  - made `depth` and `name` parameters optional. `depth` now defaults to `1` (current level only) and `name` defaults to "blueprint"
  - `depth` can now be negative, which will result in the blueprints being written from the highest z-level to the lowest. Before, blueprints were always written from the lowest z-level to the highest.
  - added the `--cursor` option to set the starting coordinate for the generated blueprints. A game cursor is no longer necessary if this option is used.
- *devel/annc-monitor*: added `report enable|disable` subcommand to filter combat reports
- *embark-assistant*: slightly improved performance of surveying and improved code a little
- *gui/advfort*: added workshop name to workshop UI
- ***quickfort***:
  - the Dreamfort blueprint set can now be comfortably built in a 1x1 embark
  - added the `--cursor` option for running a blueprint at specific coordinates instead of starting at the game cursor position
  - added more helpful error messages for invalid modeline markers

  - added support for extra space characters in blueprints
  - added a warning when an invalid alias is encountered instead of silently ignoring it
  - made more quiet when the `--quiet` parameter is specified
- *setfps*: improved error handling
- stonesense: sped up startup time
- *tweak* hide-priority: changed so that priorities stay hidden (or visible) when exiting and re-entering the designations menu
- *unretire-anyone*: the historical figure selection list now includes the `SYN_NAME` (necromancer, vampire, etc) of figures where applicable

API

- Added `dfhack.maps.getPlantAtTile(x, y, z)` and `dfhack.maps.getPlantAtTile(pos)`, and updated `dfhack.gui.getSelectedPlant()` to use it
- Added `dfhack.units.teleport(unit, pos)`

Internals

- Room definitions and extents are now created for abstract buildings so callers don't have to initialize the room structure themselves
- The DFHack test harness is now much easier to use for iterative development. Configuration can now be specified on the commandline, there are more test filter options, and the test harness can now easily rerun tests that have been run before.
- The `test/main` command to invoke the test harness has been renamed to just `test`
- Unit tests can now use `delay_until(predicate_fn, timeout_frames)` to delay until a condition is met
- Unit tests must now match any output expected to be printed via `dfhack.printerr()`
- Unit tests now support fortress mode (allowing tests that require a fortress map to be loaded) - note that these tests are skipped by continuous integration for now, pending a suitable test fortress

Lua

- new library: `argparse` is a collection of commandline argument processing functions
- **new string utility functions:**
  - `string:wrap(width)` wraps a string at space-separated word boundaries
  - `string:trim()` removes whitespace characters from the beginning and end of the string
  - `string:split(delimiter, plain)` splits a string with the given delimiter and returns a table of substrings. if `plain` is specified and set to `true`, `delimiter` is interpreted as a literal string instead of as a pattern (the default)
- new utility function: `utils.normalizePath()`: normalizes directory slashes across platoforms to `/` and coaleses adjacent directory separators
- *reveal*: now exposes `unhideFlood(pos)` functionality to Lua

- *xlsxreader*: added Lua class wrappers for the xlsxreader plugin API
- **`argparse.processArgsGetopt()` (previously `utils.processArgsGetopt()`):**
  - now returns negative numbers (e.g. `-10`) in the list of positional parameters instead of treating it as an option string equivalent to `-1 -0`
  - now properly handles `--` like GNU `getopt` as a marker to treat all further parameters as non-options
  - now detects when required arguments to long-form options are missing
- `gui.dwarfmode`: new function: `enterSidebarMode(sidebar_mode, max_esc)` which uses keypresses to get into the specified sidebar mode from whatever the current screen is
- `gui.Painter`: fixed error when calling `viewport()` method

Structures

- Identified remaining rhythm beat enum values
- `ui_advmode.interactions`: identified some fields related to party members
- `ui_advmode_menu`: identified several enum items
- **`ui_advmode`:**
  - identified several fields
  - renamed `wait` to `rest_mode` and changed to an enum with correct values
- `viewscreen_legendsst.cur_page`: added missing `Books` enum item, which fixes some other values

Documentation

- Added more client library implementations to the *remote interface docs*

9.4.5 DFHack 0.47.05-r1

Fixes

- *confirm*: stopped exposing alternate names when convicting units
- *prospect*: improved pre embark rough estimates, particularly for small clusters

Misc Improvements

- *autohauler*: allowed the `Alchemist` labor to be enabled in *manipulator* and other labor screens so it can be used for its intended purpose of flagging that no hauling labors should be assigned to a dwarf. Before, the only way to set the flag was to use an external program like Dwarf Therapist.
- *embark-assistant*: slightly improved performance of surveying
- *gui/no-dfhack-init*: clarified how to dismiss dialog that displays when no `dfhack.init` file is found
- ***quickfort*:**
  - Dreamfort blueprint set improvements: significant refinements across the entire blueprint set. Dreamfort is now much faster, much more efficient, and much easier to use. The checklist now includes a mini-walkthrough for quick reference. The spreadsheet now also includes embark profile suggestions

  - added aliases for configuring masterwork and artifact core quality for all stockpile categories that have them; made it possible to take from multiple stockpiles in the `quantumstop` alias
  - an active cursor is no longer required for running #notes blueprints (like the dreamfort walkthrough)
  - you can now be in any mode with an active cursor when running `#query` blueprints (before you could only be in a few "approved" modes, like look, query, or place)
  - refined `#query` blueprint sanity checks: cursor should still be on target tile at end of configuration, and it's ok for the screen ID to change if you are destroying (or canceling destruction of) a building
  - now reports how many work orders were added when generating manager orders from blueprints in the gui dialog
  - added `--dry-run` option to process blueprints but not change any game state
  - you can now specify the number of desired barrels, bins, and wheelbarrows for individual stockpiles when placing them
  - `quickfort orders` on a `#place` blueprint will now enqueue manager orders for barrels, bins, or wheelbarrows that are explicitly set in the blueprint.
  - you can now add alias definitions directly to your blueprint files instead of having to put them in a separate aliases.txt file. makes sharing blueprints with custom alias definitions much easier.

Structures

- Identified scattered enum values (some rhythm beats, a couple of corruption unit thoughts, and a few language name categories)
- `viewscreen_loadgamest`: renamed `cur_step` enumeration to match style of `viewscreen_adopt_regionst` and `viewscreen_savegamest`
- `viewscreen_savegamest`: identified `cur_step` enumeration

Documentation

- *digfort*: added deprecation warnings - digfort has been replaced by *quickfort*
- *fortplan*: added deprecation warnings - fortplan has been replaced by *quickfort*

9.4.6 DFHack 0.47.05-beta1

Fixes

- *embark-assistant*: fixed bug in soil depth determination for ocean tiles
- *orders*: don't crash when importing orders with malformed JSON
- *quickfort*: raw numeric *Dig priorities* (e.g. `3`, which is a valid shorthand for `d3`) now works when used in .xlsx blueprints

Misc Improvements

- *quickfort*: new commandline options for setting the initial state of the gui dialog. for example: `quickfort gui -l dreamfort notes` will start the dialog filtered for the dreamfort walkthrough blueprints

Structures

- Dropped support for 0.47.03-0.47.04

9.4.7 DFHack 0.47.04-r5

New Scripts

- *gui/quickfort*: fast access to the quickfort interactive dialog
- *workorder-recheck*: resets the selected work order to the `Checking` state

Fixes

- ***embark-assistant*:**
  - fixed order of factors when calculating min temperature
  - improved performance of surveying
- ***quickfort*:**
  - fixed eventual crashes when creating zones
  - fixed library aliases for tallow and iron, copper, and steel weapons
  - zones are now created in the active state by default
  - solve rare crash when changing UI modes
- *search*: fixed crash when searching the `k` sidebar and navigating to another tile with certain keys, like < or >
- *seedwatch*: fixed an issue where the plugin would disable itself on map load
- *stockflow*: fixed `j` character being intercepted when naming stockpiles
- *stockpiles*: no longer outputs hotkey help text beneath *stockflow* hotkey help text

Misc Improvements

- Lua label widgets (used in all standard message boxes) are now scrollable with Up/Down/PgUp/PgDn keys
- *autofarm*: now fallows farms if all plants have reached the desired count
- ***buildingplan*:**
  - added ability to set global settings from the console, e.g. `buildingplan set boulders false`
  - added "enable all" option for buildingplan (so you don't have to enable all building types individually). This setting is not persisted (just like quickfort_mode is not persisted), but it can be set from onMapLoad.init
  - modified `Planning Mode` status in the UI to show whether the plugin is in quickfort mode, "enable all" mode, or whether just the building type is enabled.
- ***quickfort*:**
  - Dreamfort blueprint set improvements: added a streamlined checklist for all required dreamfort commands and gave names to stockpiles, levers, bridges, and zones

  - added aliases for bronze weapons and armor
  - added alias for tradeable crafts
  - new blueprint mode: `#ignore`, useful for scratch space or personal notes
  - implement `{Empty}` keycode for use in quickfort aliases; useful for defining blank-by-default alias values
  - more flexible commandline parsing allowing for more natural parameter ordering (e.g. where you used to have to write `quickfort list dreamfort -l` you can now write `quickfort list -l dreamfort`)
  - print out blueprint names that a `#meta` blueprint is applying so it's easier to understand what meta blueprints are doing
  - whitespace is now allowed between a marker name and the opening parenthesis in blueprint modelines. for example, `#dig start (5; 5)` is now valid (you used to be required to write `#dig start(5; 5)`)

Lua

- `dfhack.run_command()`: changed to interface directly with the console when possible, which allows interactive commands and commands that detect the console encoding to work properly
- `processArgsGetopt()` added to utils.lua, providing a callback interface for parameter parsing and getopt-like flexibility for parameter ordering and combination (see docs in `library/lua/utils.lua` and `library/lua/3rdparty/alt_getopt.lua` for details).

Structures

- `job`: identified `order_id` field

Documentation

- Added documentation for Lua's `dfhack.run_command()` and variants

9.4.8 DFHack 0.47.04-r4

New Scripts

- *fix/corrupt-equipment*: fixes some military equipment-related corruption issues that can cause DF crashes

Fixes

- Fixed an issue on some Linux systems where DFHack installed through a package manager would attempt to write files to a non-writable folder (notably when running *exportlegends* or *gui/autogems*)
- *adaptation*: fixed handling of units with no cave adaptation suffered yet
- *assign-goals*: fixed error preventing new goals from being created
- *assign-preferences*: fixed handling of preferences for flour
- ***buildingplan*:**

  - fixed an issue preventing artifacts from being matched when the maximum item quality is set to `artifacts`
  - stopped erroneously matching items to buildings while the game is paused
  - fixed a crash when pressing 0 while having a noble room selected
- *deathcause*: fixed an error when inspecting certain corpses
- *dwarfmonitor*: fixed a crash when opening the `prefs` screen if units have vague preferences
- *dwarfvet*: fixed a crash that could occur when discharging patients
- ***embark-assistant*:**
  - fixed an issue causing incursion resource matching (e.g. sand/clay) to skip some tiles if those resources were provided only through incursions
  - corrected river size determination by performing it at the MLT level rather than the world tile level
- ***quickfort*:**
  - fixed handling of modifier keys (e.g. `{Ctrl}` or `{Alt}`) in query blueprints
  - fixed misconfiguration of nest boxes, hives, and slabs that were preventing them from being built from build blueprints
  - fixed valid placement detection for floor hatches, floor grates, and floor bars (they were erroneously being rejected from open spaces and staircase tops)
  - fixed query blueprint statistics being added to the wrong metric when both a query and a zone blueprint are run by the same meta blueprint
  - added missing blueprint labels in gui dialog list
  - fixed occupancy settings for extent-based structures so that stockpiles can be placed within other stockpiles (e.g. in a checkerboard or bullseye pattern)
- *search*: fixed an issue where search options might not display if screens were destroyed and recreated programmatically (e.g. with *quickfort*)
- *unsuspend*: now leaves buildingplan-managed buildings alone and doesn't unsuspend underwater tasks
- *workflow*: fixed an error when creating constraints on "mill plants" jobs and some other plant-related jobs
- *zone*: fixed an issue causing the `enumnick` subcommand to run when attempting to run `assign`, `unassign`, or `slaughter`

Misc Improvements

- ***buildingplan*:**
  - added support for all buildings, furniture, and constructions (except for instruments)
  - added support for respecting building job_item filters when matching items, so you can set your own programmatic filters for buildings before submitting them to buildingplan
  - changed default filter setting for max quality from `artifact` to `masterwork`
  - changed min quality adjustment hotkeys from 'qw' to 'QW' to avoid conflict with existing hotkeys for setting roller speed - also changed max quality adjustment hotkeys from 'QW' to 'AS' to make room for the min quality hotkey changes
  - added a new global settings page accessible via the `G` hotkey when on any building build screen; `Quickfort Mode` toggle for legacy Python Quickfort has been moved to this page

  - added new global settings for whether generic building materials should match blocks, boulders, logs, and/or bars - defaults are everything but bars
- *devel/export-dt-ini*: updated for Dwarf Therapist 41.2.0
- *embark-assistant*: split the lair types displayed on the local map into mound, burrow, and lair
- *gui/advfort*: added support for linking to hatches and pressure plates with mechanisms
- *modtools/add-syndrome*: added support for specifying syndrome IDs instead of names
- *probe*: added more output for designations and tile occupancy
- ***quickfort*:**
  - The Dreamfort sample blueprints now have complete walkthroughs for each fort level and importable orders that automate basic fort stock management
  - added more blueprints to the blueprints library: several bedroom layouts, the Saracen Crypts, and the complete fortress example from Python Quickfort: TheQuickFortress
  - query blueprint aliases can now accept parameters for dynamic expansion - see dfhack-config/quickfort/aliases.txt for details
  - alias names can now include dashes and underscores (in addition to letters and numbers)
  - improved speed of first call to `quickfort list` significantly, especially for large blueprint libraries
  - added `query_unsafe` setting to disable query blueprint error checking - useful for query blueprints that send unusual key sequences
  - added support for bookcases, display cases, and offering places (altars)
  - added configuration support for zone pit/pond, gather, and hospital sub-menus in zone blueprints
  - removed `buildings_use_blocks` setting and replaced it with more flexible functionality in *buildingplan*
  - added support for creating uninitialized stockpiles with `c`

API

- *buildingplan*: added Lua interface API
- `Buildings::setSize()`: changed to reuse existing extents when possible
- `dfhack.job.isSuitableMaterial()`: added an item type parameter so the `non_economic` flag can be properly handled (it was being matched for all item types instead of just boulders)

Lua

- `utils.addressof()`: fixed for raw userdata

Structures

- `building_extents_type`: new enum, used for `building_extents.extents`
- `world_mountain_peak`: new struct (was previously inline) - used in `world_data.mountain_peaks`

Documentation

- *Quickfort Keystroke Alias Guide*: alias syntax and alias standard library documentation for *quickfort* blueprints
- *Blueprint Library Index*: overview of the quickfort blueprint library

9.4.9 DFHack 0.47.04-r3

New Plugins

- *xlsxreader*: provides an API for Lua scripts to read Excel spreadsheets

New Scripts

- *quickfort*: DFHack-native implementation of quickfort with many new features and integrations - see the *Quickfort Blueprint Editing Guide* for details
- *timestream*: controls the speed of the calendar and creatures
- *uniform-unstick*: prompts units to reevaluate their uniform, by removing/dropping potentially conflicting worn items

Fixes

- *ban-cooking*: fixed an error in several subcommands
- *buildingplan*: fixed handling of buildings that require buckets
- *getplants*: fixed a crash that could occur on some maps
- *search*: fixed an issue causing item counts on the trade screen to display inconsistently when searching
- ***stockpiles*:**
  - fixed a crash when loading food stockpiles
  - fixed an error when saving furniture stockpiles

Misc Improvements

- ***createitem*:**
  - added support for plant growths (fruit, berries, leaves, etc.)
  - added an `inspect` subcommand to print the item and material tokens of existing items, which can be used to create additional matching items
- *embark-assistant*: added support for searching for taller waterfalls (up to 50 z-levels tall)
- *search*: added support for searching for names containing non-ASCII characters using their ASCII equivalents
- *stocks*: added support for searching for items containing non-ASCII characters using their ASCII equivalents
- *unretire-anyone*: made undead creature names appear in the historical figure list
- ***zone*:**
  - added an `enumnick` subcommand to assign enumerated nicknames (e.g "Hen 1", "Hen 2"...)
  - added slaughter indication to `uinfo` output

API

- Added `DFHack::to_search_normalized()` (Lua: `dfhack.toSearchNormalized()`) to convert non-ASCII alphabetic characters to their ASCII equivalents

Structures

- `history_event_masterpiece_createdst`: fixed alignment, including subclasses, and identified `skill_at_time`
- `item_body_component`: fixed some alignment issues and identified some fields (also applies to subclasses like `item_corpsest`)
- `stockpile_settings`: removed `furniture.sand_bags` (no longer present)

Documentation

- Fixed syntax highlighting of most code blocks to use the appropriate language (or no language) instead of Python

9.4.10 DFHack 0.47.04-r2

New Scripts

- *animal-control*: helps manage the butchery and gelding of animals
- *devel/kill-hf*: kills a historical figure
- *geld*: gelds or ungelds animals
- *list-agreements*: lists all guildhall and temple agreements
- *list-waves*: displays migration wave information for citizens/units
- *ungeld*: ungelds animals (wrapper around *geld*)

New Tweaks

- *tweak* do-job-now: adds a job priority toggle to the jobs list
- *tweak* reaction-gloves: adds an option to make reactions produce gloves in sets with correct handedness

Fixes

- Fixed a segfault when attempting to start a headless session with a graphical PRINT_MODE setting
- Fixed an issue with the macOS launcher failing to un-quarantine some files
- Fixed `Units::isEggLayer`, `Units::isGrazer`, `Units::isMilkable`, `Units::isTrainableHunting`, `Units::isTrainableWar`, and `Units::isTamable` ignoring the unit's caste
- Linux: fixed `dfhack.getDFPath()` (Lua) and `Process::getPath()` (C++) to always return the DF root path, even if the working directory has changed
- ***digfort***:
  - fixed y-line tracking when .csv files contain lines with only commas

  - fixed an issue causing blueprints touching the southern or eastern edges of the map to be rejected (northern and western edges were already allowed). This allows blueprints that span the entire embark area.

- *embark-assistant*: fixed a couple of incursion handling bugs.
- *embark-skills*: fixed an issue with structures causing the `points` option to do nothing
- ***exportlegends*:**
  - fixed an issue where two different `<reason>` tags could be included in a `<historical_event>`
  - stopped including some tags with `-1` values which don't provide useful information
- *getplants*: fixed issues causing plants to be collected even if they have no growths (or unripe growths)
- *gui/advfort*: fixed "operate pump" job
- *gui/load-screen*: fixed an issue causing longer timezones to be cut off
- ***labormanager*:**
  - fixed handling of new jobs in 0.47
  - fixed an issue preventing custom furnaces from being built
- ***modtools/moddable-gods*:**
  - fixed an error when creating the historical figure
  - removed unused `-domain` and `-description` arguments
  - made `-depictedAs` argument work
- ***names*:**
  - fixed an error preventing the script from working
  - fixed an issue causing renamed units to display their old name in legends mode and some other places
- *pref-adjust*: fixed some compatibility issues and a potential crash
- ***remotefortressreader*:**
  - fixed a couple crashes that could result from decoding invalid enum items (`site_realization_building_type` and `improvement_type`)
  - fixed an issue that could cause block coordinates to be incorrect
- *rendermax*: fixed a hang that could occur when enabling some renderers, notably on Linux
- **stonesense:**
  - fixed a crash when launching Stonesense
  - fixed some issues that could cause the splash screen to hang

Misc Improvements

- Linux/macOS: Added console keybindings for deleting words (Alt+Backspace and Alt+d in most terminals)
- ***add-recipe*:**
  - added tool recipes (minecarts, wheelbarrows, stepladders, etc.)
  - added a command explanation or error message when entering an invalid command
- *armoks-blessing*: added adjustments to values and needs

- ***blueprint*:**
  - now writes blueprints to the `blueprints/` subfolder instead of the df root folder
  - now automatically creates folder trees when organizing blueprints into subfolders (e.g. `blueprint 30 30 1 rooms/dining dig` will create the file `blueprints/rooms/dining-dig.csv`); previously it would fail if the `blueprints/rooms/` directory didn't already exist
- *confirm*: added a confirmation dialog for convicting dwarves of crimes
- *devel/query*: added many new query options
- ***digfort*:**
  - handled double quotes (") at the start of a string, allowing .csv files exported from spreadsheets to work without manual modification
  - documented that removing ramps, cutting trees, and gathering plants are indeed supported
  - added a `force` option to truncate blueprints if the full blueprint would extend off the edge of the map
- ***dwarf-op*:**
  - added ability to select dwarves based on migration wave
  - added ability to protect dwarves based on symbols in their custom professions
- ***exportlegends*:**
  - changed some flags to be represented by self-closing tags instead of true/false strings (e.g. `<is_volcano/>`) - note that this may require changes to other XML-parsing utilities
  - changed some enum values from numbers to their string representations
  - added ability to save all files to a subfolder, named after the region folder and date by default
- *gui/advfort*: added support for specifying the entity used to determine available resources
- *gui/gm-editor*: added support for automatically following ref-targets when pressing the `i` key
- *manipulator*: added a new column option to display units' goals
- *modtools/moddable-gods*: added support for `neuter` gender
- ***pref-adjust*:**
  - added support for adjusting just the selected dwarf
  - added a new `goth` profile
- *remove-stress*: added a `-value` argument to enable setting stress level directly
- *workorder*: changed default frequency from "Daily" to "OneTime"

API

- Added `Filesystem::mkdir_recursive`
- Extended `Filesystem::listdir_recursive` to optionally make returned filenames relative to the start directory
- `Units`: added goal-related functions: `getGoalType()`, `getGoalName()`, `isGoalAchieved()`

Internals

- Added support for splitting scripts into multiple files in the `scripts/internal` folder without polluting the output of *ls*

Lua

- Added a `ref_target` field to primitive field references, corresponding to the `ref-target` XML attribute
- Made `dfhack.units.getRaceNameById()`, `dfhack.units.getRaceBabyNameById()`, and `dfhack.units.getRaceChildNameById()` available to Lua

Ruby

- Updated `item_find` and `building_find` to use centralized logic that works on more screens

Structures

- Added a new `<df-other-vectors-type>`, which allows `world.*.other` collections of vectors to use the correct subtypes for items
- `creature_raw`: renamed `gender` to `sex` to match the field in `unit`, which is more frequently used
- `crime`: identified `witnesses`, which contains the data held by the old field named `reports`
- `intrigue`: new type (split out from `historical_figure_relationships`)
- `items_other_id`: removed `BAD`, and by extension, `world.items.other.BAD`, which was overlapping with `world.items.bad`
- `job_type`: added job types new to 0.47
- `plant_raw`: material_defs now contains arrays rather than loose fields
- `pronoun_type`: new enum (previously documented in field comments)
- `setup_character_info`: fixed a couple alignment issues (needed by *embark-skills*)
- `ui_advmode_menu`: identified some new enum items

Documentation

- Added some new dev-facing pages, including dedicated pages about the remote API, memory research, and documentation
- Expanded the installation guide
- Made a couple theme adjustments

9.4.11 DFHack 0.47.04-r1

Fixes

- Fixed a crash in `find()` for some types when no world is loaded
- Fixed translation of certain types of in-game names

- *autogems*: fixed an issue with binned gems being ignored in linked stockpiles
- *catsplosion*: fixed error when handling races with only one caste (e.g. harpies)
- *exportlegends*: fixed error when exporting maps
- *spawnunit*: fixed an error when forwarding some arguments but not a location to *modtools/create-unit*
- *stocks*: fixed display of book titles
- *tweak* embark-profile-name: fixed handling of the native shift+space key

Misc Improvements

- ***exportlegends*:**
  - made interaction export more robust and human-readable
  - removed empty `<item_subtype>` and `<claims>` tags
- *getplants*: added switches for designations for farming seeds and for max number designated per plant
- *manipulator*: added intrigue to displayed skills
- ***modtools/create-unit*:**
  - added `-equip` option to equip created units
  - added `-skills` option to give skills to units
  - added `-profession` and `-customProfession` options to adjust unit professions
- *search*: added support for the fortress mode justice screen
- `dfhack.init-example`: enabled *autodump*

API

- Added `Items::getBookTitle` to get titles of books. Catches titles buried in improvements, unlike getDescription.

Lua

- `pairs()` now returns available class methods for DF types

Structures

- Added globals: `cur_rain`, `cur_rain_counter`, `cur_snow`, `cur_snow_counter`, `weathertimer`, `jobvalue`, `jobvalue_setter`, `interactitem`, `interactinvslot`, `handleannounce`, `preserveannounce`, `updatelightstate`
- `agreement_details_data_plot_sabotage`: new struct type, along with related `agreement_details_type.PlotSabotage`
- `architectural_element`: new enum
- `battlefield`: new struct type
- `breed`: new struct type
- `creature_handler`: identified vmethods

- `crime`: removed fields of `reports` that are no longer present
- `dance_form`: identified most fields
- `history_event_context`: identified fields
- `identity_type`: new enum
- `identity`: renamed `civ` to `entity_id`, identified `type`
- `image_set`: new struct type
- `interrogation_report`: new struct type
- `itemdef_flags`: new enum, with `GENERATED` flag
- `justification`: new enum
- `lever_target_type`: identified `LeverMechanism` and `TargetMechanism` values
- `musical_form`: identified fields, including some renames. Also identified fields in `scale` and `rhythm`
- `region_weather`: new struct type
- `squad_order_cause_trouble_for_entityst`: identified fields
- `unit_thought_type`: added several new thought types
- `viewscreen_workquota_detailsst`: identified fields

9.4.12 DFHack 0.47.04-beta1

New Scripts

- *color-schemes*: manages color schemes
- *devel/print-event*: prints the description of an event by ID or index
- *gui/color-schemes*: an in-game interface for *color-schemes*
- *light-aquifers-only*: changes heavy aquifers to light aquifers
- *on-new-fortress*: runs DFHack commands only in a new fortress
- *once-per-save*: runs DFHack commands unless already run in the current save
- *resurrect-adv*: brings your adventurer back to life
- *reveal-hidden-units*: exposes all sneaking units
- *workorder*: allows queuing manager jobs; smart about shear and milk creature jobs

Fixes

- Fixed a crash when starting DFHack in headless mode with no terminal
- *devel/visualize-structure*: fixed padding detection for globals
- ***exportlegends*:**
  - added UTF-8 encoding and XML escaping for more fields
  - added checking for unhandled structures to avoid generating invalid XML
  - fixed missing fields in `history_event_assume_identityst` export

- ***full-heal*:**
  - when resurrected by specifying a corpse, units now appear at the location of the corpse rather than their location of death
  - resurrected units now have their tile occupancy set (and are placed in the prone position to facilitate this)

Misc Improvements

- Added "bit" suffix to downloads (e.g. 64-bit)
- **Tests:**
  - moved from DF folder to hack/scripts folder, and disabled installation by default
  - made test runner script more flexible
- *devel/export-dt-ini*: updated some field names for DT for 0.47
- *devel/visualize-structure*: added human-readable lengths to containers
- *dfhack-run*: added color output support
- ***embark-assistant*:**
  - updated embark aquifer info to show all aquifer kinds present
  - added neighbor display, including kobolds (SKULKING) and necro tower count
  - updated aquifer search criteria to handle the new variation
  - added search criteria for embark initial tree cover
  - added search criteria for necro tower count, neighbor civ count, and specific neighbors. Should handle additional entities, but not tested
- ***exportlegends*:**
  - added evilness and force IDs to regions
  - added profession and weapon info to relevant entities
  - added support for many new history events in 0.47
  - added historical event relationships and supplementary data
- ***full-heal*:**
  - made resurrection produce a historical event viewable in Legends mode
  - made error messages more explanatory
- *install-info*: added DFHack build ID to report
- *modtools/create-item*: added `-matchingGloves` and `-matchingShoes` arguments
- ***modtools/create-unit*:**
  - added `-duration` argument to make the unit vanish after some time
  - added `-locationRange` argument to allow spawning in a random position within a defined area
  - added `-locationType` argument to specify the type of location to spawn in

Internals

- Added separate changelogs in the scripts and df-structures repos
- Improved support for tagged unions, allowing tools to access union fields more safely
- Moved `reversing` scripts to df_misc repo

Structures

- Added an XML schema for validating df-structures syntax
- Added `divination_set_next_id` and `image_set_next_id` globals
- `activity_entry_type`: new enum type
- `adventure_optionst`: identified many vmethods
- `agreement_details`: identified most fields of most sub-structs
- `artifact_claim`: identified several fields
- `artifact_record`: identified several fields
- `caste_raw_flags`: renamed and identified many flags to match information from Toady
- `creature_raw_flags`: renamed and identified many flags to match information from Toady
- `crime_type`: new enum type
- `dfhack_room_quality_level`: added enum attributes for names of rooms of each quality
- `entity_site_link_type`: new enum type
- `export_map_type`: new enum type
- `historical_entity.flags`: identified several flags
- `historical_entity.relations`: renamed from `unknown1b` and identified several fields
- `historical_figure.vague_relationships`: identified
- `historical_figure_info.known_info`: renamed from `secret`, identified some fields
- `historical_figure`: renamed `unit_id2` to `nemesis_id`
- `history_event_circumstance_info`: new struct type (and changed several `history_event` subclasses to use this)
- `history_event_reason_info`: new struct type (and changed several `history_event` subclasses to use this)
- `honors_type`: identified several fields
- `interaction_effect_create_itemst`: new struct type
- `interaction_effect_summon_unitst`: new struct type
- `item`: identified several vmethods
- `layer_type`: new enum type
- `plant.damage_flags`: added `is_dead`
- `plot_role_type`: new enum type
- `plot_strategy_type`: new enum type

- `relationship_event_supplement`: new struct type
- `relationship_event`: new struct type
- `specific_ref`: moved union data to `data` field
- `ui_look_list`: moved union fields to `data` and renamed to match `type` enum
- `ui_sidebar_menus.location`: added new profession-related fields, renamed and fixed types of deity-related fields
- `ui_sidebar_mode`: added `ZonesLocationInfo`
- `unit_action`: rearranged as tagged union with new sub-types; existing code should be compatible
- `vague_relationship_type`: new enum type
- `vermin_flags`: identified `is_roaming_colony`
- `viewscreen_justicest`: identified interrogation-related fields
- `world_data.field_battles`: identified and named several fields

9.4.13 DFHack 0.47.03-beta1

New Scripts

- *devel/sc*: checks size of structures
- *devel/visualize-structure*: displays the raw memory of a structure

Fixes

- *adv-max-skills*: fixed for 0.47
- ***deep-embark***:
  - prevented running in non-fortress modes
  - ensured that only the newest wagon is deconstructed
- ***full-heal***:
  - fixed issues with removing corpses
  - fixed resurrection for non-historical figures
- *modtools/create-unit*: added handling for arena tame setting
- *teleport*: fixed setting new tile occupancy

Misc Improvements

- ***deep-embark***:
  - improved support for using directly from the DFHack console
  - added a `-clear` option to cancel
- ***exportlegends***:
  - added identity information

  - added creature raw names and flags
- *gui/prerelease-warning*: updated links and information about nightly builds
- *modtools/syndrome-trigger*: enabled simultaneous use of `-synclass` and `-syndrome`
- *repeat*: added `-list` option

Structures

- Dropped support for 0.44.12-0.47.02
- `abstract_building_type`: added types (and subclasses) new to 0.47
- `agreement_details_type`: added enum
- `agreement_details`: added struct type (and many associated data types)
- `agreement_party`: added struct type
- `announcement_type`: added types new to 0.47
- `artifact_claim_type`: added enum
- `artifact_claim`: added struct type
- `breath_attack_type`: added `SHARP_ROCK`
- `building_offering_placest`: new class
- `building_type`: added `OfferingPlace`
- `caste_raw_flags`: renamed many items to match DF names
- `creature_interaction_effect`: added subclasses new to 0.47
- **`creature_raw_flags`:**
  - identified several more items
  - renamed many items to match DF names
- `d_init`: added settings new to 0.47
- `entity_name_type`: added `MERCHANT_COMPANY`, `CRAFT_GUILD`
- `entity_position_responsibility`: added values new to 0.47
- `fortress_type`: added enum
- `general_ref_type`: added `UNIT_INTERROGATEE`
- `ghost_type`: added `None` value
- `goal_type`: added goals types new to 0.47
- `histfig_site_link`: added subclasses new to 0.47
- `history_event_collection`: added subtypes new to 0.47
- `history_event_context`: added lots of new fields
- **`history_event_reason`:**
  - added captions for all items
  - added items new to 0.47

- history_event_type: added types for events new to 0.47, as well as corresponding history_event subclasses (too many to list here)
- honors_type: added struct type
- interaction_effect: added subtypes new to 0.47
- interaction_source_experimentst: added class type
- interaction_source_usage_hint: added values new to 0.47
- interface_key: added items for keys new to 0.47
- job_skill: added INTRIGUE, RIDING
- lair_type: added enum
- monument_type: added enum
- next_global_id: added enum
- poetic_form_action: added Beseech
- setup_character_info: expanded significantly in 0.47
- text_system: added layout for struct
- tile_occupancy: added varied_heavy_aquifer
- tool_uses: added items: PLACE_OFFERING, DIVINATION, GAMES_OF_CHANCE
- viewscreen_counterintelligencest: new class (only layout identified so far)

9.4.14 DFHack 0.44.12-r3

New Plugins

- *autoclothing*: automatically manage clothing work orders
- *autofarm*: replaces the previous Ruby script of the same name, with some fixes
- *map-render*: allows programmatically rendering sections of the map that are off-screen
- *tailor*: automatically manages keeping your dorfs clothed

New Scripts

- *assign-attributes*: changes the attributes of a unit
- *assign-beliefs*: changes the beliefs of a unit
- *assign-facets*: changes the facets (traits) of a unit
- *assign-goals*: changes the goals of a unit
- *assign-preferences*: changes the preferences of a unit
- *assign-profile*: sets a dwarf's characteristics according to a predefined profile
- *assign-skills*: changes the skills of a unit
- *combat-harden*: sets a unit's combat-hardened value to a given percent
- *deep-embark*: allows embarking underground
- *devel/find-twbt*: finds a TWBT-related offset needed by the new *map-render* plugin

- *dwarf-op*: optimizes dwarves for fort-mode work; makes managing labors easier
- *forget-dead-body*: removes emotions associated with seeing a dead body
- *gui/create-tree*: creates a tree at the selected tile
- *linger*: takes over your killer in adventure mode
- *modtools/create-tree*: creates a tree
- *modtools/pref-edit*: add, remove, or edit the preferences of a unit
- *modtools/set-belief*: changes the beliefs (values) of units
- *modtools/set-need*: sets and edits unit needs
- *modtools/set-personality*: changes the personality of units
- *modtools/spawn-liquid*: spawns water or lava at the specified coordinates
- *set-orientation*: edits a unit's orientation
- *unretire-anyone*: turns any historical figure into a playable adventurer

Fixes

- Fixed a crash in the macOS/Linux console when the prompt was wider than the screen width
- Fixed inconsistent results from `Units::isGay` for asexual units
- Fixed some cases where Lua filtered lists would not properly intercept keys, potentially triggering other actions on the same screen
- ***autofarm*:**
  - fixed biome detection to properly determine crop assignments on surface farms
  - reimplemented as a C++ plugin to make proper biome detection possible
- *bodyswap*: fixed companion list not being updated often enough
- *cxxrandom*: removed some extraneous debug information
- *digfort*: now accounts for z-level changes when calculating maximum y dimension
- ***embark-assistant*:**
  - fixed bug causing crash on worlds without generated metals (as well as pruning vectors as originally intended).
  - fixed bug causing mineral matching to fail to cut off at the magma sea, reporting presence of things that aren't (like DF does currently).
  - fixed bug causing half of the river tiles not to be recognized.
  - added logic to detect some river tiles DF doesn't generate data for (but are definitely present).
- *eventful*: fixed invalid building ID in some building events
- *exportlegends*: now escapes special characters in names properly
- *getplants*: fixed designation of plants out of season (note that picked plants are still designated incorrectly)
- *gui/autogems*: fixed error when no world is loaded
- ***gui/companion-order*:**
  - fixed error when resetting group leaders

  - `leave` now properly removes companion links
- *gui/create-item*: fixed module support - can now be used from other scripts
- ***gui/stamper*:**
  - stopped "invert" from resetting the designation type
  - switched to using DF's designation keybindings instead of custom bindings
  - fixed some typos and text overlapping
- ***modtools/create-unit*:**
  - fixed an error associating historical entities with units
  - stopped recalculating health to avoid newly-created citizens triggering a "recover wounded" job
  - fixed units created in arena mode having blank names
  - fixed units created in arena mode having the wrong race and/or interaction effects applied after creating units manually in-game
  - stopped units from spawning with extra items or skills previously selected in the arena
  - stopped setting some unneeded flags that could result in glowing creature tiles
  - set units created in adventure mode to have no family, instead of being related to the first creature in the world
- ***modtools/reaction-product-trigger*:**
  - fixed an error dealing with reactions in adventure mode
  - blocked `\\BUILDING_ID` for adventure mode reactions
  - fixed `-clear` to work without passing other unneeded arguments
- ***modtools/reaction-trigger*:**
  - fixed a bug when determining whether a command was run
  - fixed handling of `-resetPolicy`
- *mousequery*: fixed calculation of map dimensions, which was sometimes preventing scrolling the map with the mouse when TWBT was enabled
- *remotefortressreader*: fixed a crash when a unit's path has a length of 0
- stonesense: fixed crash due to wagons and other soul-less creatures
- *tame*: now sets the civ ID of tamed animals (fixes compatibility with *autobutcher*)
- *title-folder*: silenced error when `PRINT_MODE` is set to `TEXT`

Misc Improvements

- Added a note to *dfhack-run* when called with no arguments (which is usually unintentional)
- On macOS, the launcher now attempts to un-quarantine the rest of DFHack
- *bodyswap*: added arena mode support
- *combine-drinks*: added more default output, similar to *combine-plants*
- *createitem*: added a list of valid castes to the "invalid caste" error message, for convenience
- *devel/export-dt-ini*: added more size information needed by newer Dwarf Therapist versions

- *dwarfmonitor*: enabled widgets to access other scripts and plugins by switching to the core Lua context
- ***embark-assistant*:**
  - added an in-game option to activate on the embark screen
  - changed waterfall detection to look for level drop rather than just presence
  - changed matching to take incursions, i.e. parts of other biomes, into consideration when evaluating tiles. This allows for e.g. finding multiple biomes on single tile embarks.
  - changed overlay display to show when incursion surveying is incomplete
  - changed overlay display to show evil weather
  - added optional parameter "fileresult" for crude external harness automated match support
  - improved focus movement logic to go to only required world tiles, increasing speed of subsequent searches considerably
- *exportlegends*: added rivers to custom XML export
- *exterminate*: added support for a special `enemy` caste
- ***gui/gm-unit*:**
  - added support for editing:
  - added attribute editor
  - added orientation editor
  - added editor for bodies and body parts
  - added color editor
  - added belief editor
  - added personality editor
- *modtools/create-item*: documented already-existing `-quality` option
- ***modtools/create-unit*:**
  - added the ability to specify `\\LOCAL` for the fort group entity
  - now enables the default labours for adult units with CAN_LEARN.
  - now sets historical figure orientation.
  - improved speed of creating multiple units at once
  - made the script usable as a module (from other scripts)
- ***modtools/reaction-trigger*:**
  - added `-ignoreWorker`: ignores the worker when selecting the targets
  - changed the default behavior to skip inactive/dead units; added `-dontSkipInactive` to include creatures that are inactive
  - added `-range`: controls how far elligible targets can be from the workshop
  - syndromes now are applied before commands are run, not after
  - if both a command and a syndrome are given, the command only runs if the syndrome could be applied
- *mousequery*: made it more clear when features are enabled
- ***remotefortressreader*:**

  - added a basic framework for controlling and reading the menus in DF (currently only supports the building menu)
  - added support for reading item raws
  - added a check for whether or not the game is currently saving or loading, for utilities to check if it's safe to read from DF
  - added unit facing direction estimate and position within tiles
  - added unit age
  - added unit wounds
  - added tree information
  - added check for units' current jobs when calculating the direction they are facing

API

- Added new `plugin_load_data` and `plugin_save_data` events for plugins to load/save persistent data
- Added `Maps::GetBiomeType` and `Maps::GetBiomeTypeByRef` to infer biome types properly
- Added `Units::getPhysicalDescription` (note that this depends on the `unit_get_physical_description` offset, which is not yet available for all DF builds)

Internals

- Added new Persistence module
- Cut down on internal DFHack dependencies to improve build times
- Improved concurrency in event and server handlers
- Persistent data is now stored in JSON files instead of historical figures - existing data will be migrated when saving
- stonesense: fixed some OpenGL build issues on Linux

Lua

- Exposed `gui.dwarfmode.get_movement_delta` and `gui.dwarfmode.get_hotkey_target`
- `dfhack.run_command` now returns the command's return code

Ruby

- Made `unit_ishostile` consistently return a boolean

Structures

- Added `unit_get_physical_description` function offset on some platforms
- **Added/identified types:**
  - `assume_identity_mode`
  - `musical_form_purpose`

    - `musical_form_style`
    - `musical_form_pitch_style`
    - `musical_form_feature`
    - `musical_form_vocals`
    - `musical_form_melodies`
    - `musical_form_interval`
    - `unit_emotion_memory`
- `need_type`: fixed `PrayOrMeditate` typo
- `personality_facet_type`, `value_type`: added `NONE` values
- `twbt_render_map`: added for 64-bit 0.44.12 (for *map-render*)

9.4.15 DFHack 0.44.12-r2

New Plugins

- *debug*: manages runtime debug print category filtering
- *nestboxes*: automatically scan for and forbid fertile eggs incubating in a nestbox

New Scripts

- *devel/query*: searches for field names in DF objects
- *extinguish*: puts out fires
- *tame*: sets tamed/trained status of animals

Fixes

- *building-hacks*: fixed error when dealing with custom animation tables
- *devel/test-perlin*: fixed Lua error (`math.pow()`)
- *embark-assistant*: fixed crash when entering finder with a 16x16 embark selected, and added 16 to dimension choices
- *embark-skills*: fixed missing `skill_points_remaining` field
- ***full-heal*:**
    - stopped wagon resurrection
    - fixed a minor issue with post-resurrection hostility
- ***gui/companion-order*:**
    - fixed issues with printing coordinates
    - fixed issues with move command
    - fixed cheat commands (and removed "Power up", which was broken)
- *gui/gm-editor*: fixed reinterpret cast (`r`)

- *gui/pathable*: fixed error when sidebar is hidden with `Tab`
- ***labormanager*:**
  - stopped assigning labors to ineligible dwarves, pets, etc.
  - stopped assigning invalid labors
  - added support for crafting jobs that use pearl
  - fixed issues causing cleaning jobs to not be assigned
  - added support for disabling management of specific labors
- *prospect*: (also affected *embark-tools*) - fixed a crash when prospecting an unusable site (ocean, mountains, etc.) with a large default embark size in d_init.txt (e.g. 16x16)
- *siege-engine*: fixed a few Lua errors (`math.pow()`, `unit.relationship_ids`)
- *tweak*: fixed `hotkey-clear`

Misc Improvements

- *armoks-blessing*: improved documentation to list all available arguments
- ***devel/export-dt-ini*:**
  - added viewscreen offsets for DT 40.1.2
  - added item base flags offset
  - added needs offsets
- ***embark-assistant*:**
  - added match indicator display on the right ("World") map
  - changed 'c'ancel to abort find if it's under way and clear results if not, allowing use of partial surveys.
  - added Coal as a search criterion, as well as a coal indication as current embark selection info.
- ***full-heal*:**
  - added `-all`, `-all_civ` and `-all_citizens` arguments
  - added module support
  - now removes historical figure death dates and ghost data
- *growcrops*: added `all` argument to grow all crops
- *gui/load-screen*: improved documentation
- *labormanager*: now takes nature value into account when assigning jobs
- *open-legends*: added warning about risk of save corruption and improved related documentation
- *points*: added support when in `viewscreen_setupdwarfgamest` and improved error messages
- *siren*: removed break handling (relevant `misc_trait_type` was no longer used - see "Structures" section)

API

- **New debug features related to *debug* plugin:**
  - Classes (C++ only): `Signal<Signature, type_tag>`, `DebugCategory`, `DebugManager`
  - Macros: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERR`, `DBG_DECLARE`, `DBG_EXTERN`

Internals

- Added a usable unit test framework for basic tests, and a few basic tests
- Added `CMakeSettings.json` with intellisense support
- Changed `plugins/CMakeLists.custom.txt` to be ignored by git and created (if needed) at build time instead
- Core: various thread safety and memory management improvements
- Fixed CMake build dependencies for generated header files
- Fixed custom `CMAKE_CXX_FLAGS` not being passed to plugins
- Linux/macOS: changed recommended build backend from Make to Ninja (Make builds will be significantly slower now)

Lua

- `utils`: new `OrderedTable` class

Structures

- Win32: added missing vtables for `viewscreen_storesst` and `squad_order_rescue_hfst`
- `activity_event_performancest`: renamed poem as written_content_id
- `body_part_status`: identified `gelded`
- `dance_form`: named musical_form_id and musical_written_content_id
- `incident_sub6_performance.participants`: named performance_event and role_index
- **`incident_sub6_performance`:**
  - made performance_event an enum
  - named poetic_form_id, musical_form_id, and dance_form_id
- `misc_trait_type`: removed `LikesOutdoors`, `Hardened`, `TimeSinceBreak`, `OnBreak` (all unused by DF)
- `musical_form_instruments`: named minimum_required and maximum_permitted
- `musical_form`: named voices field
- `plant_tree_info`: identified `extent_east`, etc.
- `plant_tree_tile`: gave connection bits more meaningful names (e.g. `connection_east` instead of `thick_branches_1`)
- `poetic_form`: identified many fields and related enum/bitfield types
- `setup_character_info`: identified `skill_points_remaining` (for *embark-skills*)

- `ui.main`: identified `fortress_site`
- `ui.squads`: identified `kill_rect_targets_scroll`
- `ui`: fixed alignment of `main` and `squads` (fixes *tweak* hotkey-clear and DF-AI)
- **`unit_action.attack`:**
  - identified `attack_skill`
  - added `lightly_tap` and `spar_report` flags
- `unit_flags3`: identified `marked_for_gelding`
- `unit_personality`: identified `stress_drain`, `stress_boost`, `likes_outdoors`, `combat_hardened`
- `unit_storage_status`: newly identified type, stores noble holdings information (used in `viewscreen_layer_noblelistst`)
- `unit_thought_type`: added new expulsion thoughts from 0.44.12
- `viewscreen_layer_arena_creaturest`: identified item- and name-related fields
- `viewscreen_layer_militaryst`: identified `equip.assigned.assigned_items`
- `viewscreen_layer_noblelistst`: identified `storage_status` (see `unit_storage_status` type)
- **`viewscreen_new_regionst`:**
  - identified `rejection_msg`, `raw_folder`, `load_world_params`
  - changed many `int8_t` fields to `bool`
- `viewscreen_setupadventurest`: identified some nemesis and personality fields, and `page.ChooseHistfig`
- `world_data`: added `mountain_peak_flags` type, including `is_volcano`
- `world_history`: identified names and/or types of some fields
- `world_site`: identified names and/or types of some fields
- `written_content`: named poetic_form

9.4.16 DFHack 0.44.12-r1

Fixes

- Console: fixed crash when entering long commands on Linux/macOS
- Fixed special characters in *command-prompt* and other non-console in-game outputs on Linux/macOS (in tools using `df2console`)
- Removed jsoncpp's `include` and `lib` folders from DFHack builds/packages
- *die*: fixed Windows crash in exit handling
- *dwarfmonitor*, *manipulator*: fixed stress cutoffs
- *modtools/force*: fixed a bug where the help text would always be displayed and nothing useful would happen
- *ruby*: fixed calling conventions for vmethods that return strings (currently `enabler.GetKeyDisplay()`)
- *startdwarf*: fixed on 64-bit Linux

Misc Improvements

- Reduced time for designation jobs from tools like *dig* to be assigned workers
- ***embark-assistant*:**
  - Switched to standard scrolling keys, improved spacing slightly
  - Introduced scrolling of Finder search criteria, removing requirement for 46 lines to work properly (Help/Info still formatted for 46 lines).
  - Added Freezing search criterion, allowing searches for NA/Frozen/At_Least_Partial/Partial/At_Most_Partial/Never Freezing embarks.
- ***rejuvenate*:**
  - Added `-all` argument to apply to all citizens
  - Added `-force` to include units under 20 years old
  - Clarified documentation

API

- **Added to `Units` module:**
  - `getStressCategory(unit)`
  - `getStressCategoryRaw(level)`
  - `stress_cutoffs` (Lua: `getStressCutoffs()`)

Internals

- Added documentation for all RPC functions and a build-time check
- Added support for build IDs to development builds
- Changed default build architecture to 64-bit
- Use `dlsym(3)` to find vtables from libgraphics.so

Structures

- Added `start_dwarf_count` on 64-bit Linux again and fixed scanning script
- `army_controller`: added new vector from 0.44.11
- `belief_system`: new type, few fields identified
- `mental_picture`: new type, some fields identified
- **`mission_report`:**
  - new type (renamed, was `mission` before)
  - identified some fields
- `mission`: new type (used in `viewscreen_civliststs`)
- `spoils_report`: new type, most fields identified
- **`viewscreen_civliststs`:**

  - split `unk_20` into 3 pointers
  - identified new pages
  - identified new messenger-related fields
- **`viewscreen_image_creatorst`:**
  - fixed layout
  - identified many fields
- `viewscreen_reportliststst`: added new mission and spoils report-related fields (fixed layout)
- `world.languages`: identified (minimal information; whole languages stored elsewhere)
- **`world.status`:**
  - `mission_reports`: renamed, was `missions`
  - `spoils_reports`: identified
- `world.unk_131ec0`, `world.unk_131ef0`: researched layout
- `world.worldgen_status`: identified many fields
- `world`: `belief_systems`: identified

9.4.17 DFHack 0.44.12-alpha1

Fixes

- macOS: fixed `renderer` vtable address on x64 (fixes *rendermax*)
- stonesense: fixed `PLANT:DESERT_LIME:LEAF` typo

API

- Added C++-style linked list interface for DF linked lists

Structures

- Dropped 0.44.11 support
- `ui.squads`: Added fields new in 0.44.12

9.4.18 DFHack 0.44.11-beta2.1

Internals

- stonesense: fixed build

9.4.19 DFHack 0.44.11-beta2

Fixes

- Windows: Fixed console failing to initialize
- *command-prompt*: added support for commands that require a specific screen to be visible, e.g. *spotclean*
- *gui/workflow*: fixed advanced constraint menu for crafts

API

- Added `Screen::Hide` to temporarily hide screens, like *command-prompt*

9.4.20 DFHack 0.44.11-beta1

Fixes

- Fixed displayed names (from `Units::getVisibleName`) for units with identities
- Fixed potential memory leak in `Screen::show()`
- *fix/dead-units*: fixed script trying to use missing isDiplomat function

Misc Improvements

- **Console:**
  - added support for multibyte characters on Linux/macOS
  - made the console exit properly when an interactive command is active (*liquids*, *mode*, *tiletypes*)
- Linux: added automatic support for GCC sanitizers in `dfhack` script
- Made the `DFHACK_PORT` environment variable take priority over `remote-server.json`
- *dfhack-run*: added support for port specified in `remote-server.json`, to match DFHack's behavior
- *digfort*: added better map bounds checking
- ***remove-stress*:**
  - added support for `-all` as an alternative to the existing `all` argument for consistency
  - sped up significantly
  - improved output/error messages
  - now removes tantrums, depression, and obliviousness
- *ruby*: sped up handling of onupdate events

API

- Exposed `Screen::zoom()` to C++ (was Lua-only)
- New functions: `Units::isDiplomat(unit)`

Internals

- jsoncpp: updated to version 1.8.4 and switched to using a git submodule

Lua

- Added `printall_recurse` to print tables and DF references recursively. It can be also used with `^` from the *lua* interpreter.
- `gui.widgets`: `List:setChoices` clones `choices` for internal table changes

Structures

- `history_event_entity_expels_hfst`: added (new in 0.44.11)
- `history_event_site_surrenderedst`: added (new in 0.44.11)
- `history_event_type`: added `SITE_SURRENDERED`, `ENTITY_EXPELS_HF` (new in 0.44.11)
- `syndrome`: identified a few fields
- `viewscreen_civliststst`: fixed layout and identified many fields

9.4.21 DFHack 0.44.11-alpha1

Structures

- Added support for automatically sizing arrays indexed with an enum
- Dropped 0.44.10 support
- Removed stale generated CSV files and DT layouts from pre-0.43.05
- `announcement_type`: new in 0.44.11: `NEW_HOLDING`, `NEW_MARKET_LINK`
- `breath_attack_type`: added `OTHER`
- `historical_figure_info.relationships.list`: added `unk_3a-unk_3c` fields at end
- `interface_key`: added bindings new in 0.44.11
- `occupation_type`: new in 0.44.11: `MESSENGER`
- `profession`: new in 0.44.11: `MESSENGER`
- **`ui_sidebar_menus`:**
  - `unit.in_squad`: renamed to `unit.squad_list_opened`, fixed location
  - `unit`: added `expel_error` and other unknown fields new in 0.44.11
  - `hospital`: added, new in 0.44.11
  - `num_speech_tokens`, `unk_17d8`: moved out of `command_line` to fix layout on x64
- `viewscreen_civliststst`: added a few new fields (incomplete)
- `viewscreen_locationsst`: identified `edit_input`

9.4.22 DFHack 0.44.10-r2

New Plugins

- *cxxrandom*: exposes some features of the C++11 random number library to Lua

New Scripts

- *add-recipe*: adds unknown crafting recipes to the player's civ
- *gui/stamper*: allows manipulation of designations by transforms such as translations, reflections, rotations, and inversion

Fixes

- Fixed many tools incorrectly using the `dead` unit flag (they should generally check `flags2.killed` instead)
- Fixed many tools passing incorrect arguments to printf-style functions, including a few possible crashes (*change-layer*, *follow*, *forceequip*, *generated-creature-renamer*)
- Fixed several bugs in Lua scripts found by static analysis (df-luacheck)
- Fixed `-g` flag (GDB) in Linux `dfhack` script (particularly on x64)
- *autochop*, *autodump*, *autogems*, *automelt*, *autotrade*, *buildingplan*, *dwarfmonitor*, *fix-unit-occupancy*, *fortplan*, *stockflow*: fix issues with periodic tasks not working for some time after save/load cycles
- ***autogems*:**
  - stop running repeatedly when paused
  - fixed crash when furnaces are linked to same stockpiles as jeweler's workshops
- *autogems*, *fix-unit-occupancy*: stopped running when a fort isn't loaded (e.g. while embarking)
- *autounsuspend*: now skips planned buildings
- *ban-cooking*: fixed errors introduced by kitchen structure changes in 0.44.10-r1
- *buildingplan*, *fortplan*: stopped running before a world has fully loaded
- *deramp*: fixed deramp to find designations that already have jobs posted
- *dig*: fixed "Inappropriate dig square" announcements if digging job has been posted
- *fixnaked*: fixed errors due to emotion changes in 0.44
- *remove-stress*: fixed an error when running on soul-less units (e.g. with `-all`)
- *reveal*: stopped revealing tiles adjacent to tiles above open space inappropriately
- *stockpiles*: `loadstock` now sets usable and unusable weapon and armor settings
- *stocks*: stopped listing carried items under stockpiles where they were picked up from

Misc Improvements

- Added script name to messages produced by `qerror()` in Lua scripts
- Fixed an issue in around 30 scripts that could prevent edits to the files (adding valid arguments) from taking effect

- Linux: Added several new options to `dfhack` script: `--remotegdb`, `--gdbserver`, `--strace`
- *bodyswap*: improved error handling
- *buildingplan*: added max quality setting
- *caravan*: documented (new in 0.44.10-alpha1)
- *deathcause*: added "slaughtered" to descriptions
- ***embark-assistant*:**
  - changed region interaction matching to search for evil rain, syndrome rain, and reanimation rather than interaction presence (misleadingly called evil weather), reanimation, and thralling
  - gave syndrome rain and reanimation wider ranges of criterion values
- *fix/dead-units*: added a delay of around 1 month before removing units
- *fix/retrieve-units*: now re-adds units to active list to counteract *fix/dead-units*
- *item-descriptions*: fixed several grammatical errors
- ***modtools/create-unit*:**
  - added quantity argument
  - now selects a caste at random if none is specified
- ***mousequery*:**
  - migrated several features from TWBT's fork
  - added ability to drag with left/right buttons
  - added depth display for TWBT (when multilevel is enabled)
  - made shift+click jump to lower levels visible with TWBT
- *title-version*: added version to options screen too

API

- **New functions (also exposed to Lua):**
  - `Units::isKilled()`
  - `Units::isActive()`
  - `Units::isGhost()`
- Removed Vermin module (unused and obsolete)

Internals

- Added build option to generate symbols for large generated files containing df-structures metadata
- Added fallback for YouCompleteMe database lookup failures (e.g. for newly-created files)
- Improved efficiency and error handling in `stl_vsprintf` and related functions
- jsoncpp: fixed constructor with `long` on Linux

Lua

- Added `profiler` module to measure lua performance
- Enabled shift+cursor movement in WorkshopOverlay-derived screens

Structures

- `incident_sub6_performance`: identified some fields
- `item_body_component`: fixed location of `corpse_flags`
- `job_handler`: fixed static array layout
- `job_type`: added `is_designation` attribute
- `unit_flags1`: renamed `dead` to `inactive` to better reflect its use
- `unit_personality`: fixed location of `current_focus` and `undistracted_focus`
- `unit_thought_type`: added `SawDeadBody` (new in 0.44.10)

9.4.23 DFHack 0.44.10-r1

New Scripts

- *bodyswap*: shifts player control over to another unit in adventure mode

New Tweaks

- *tweak* kitchen-prefs-all: adds an option to toggle cook/brew for all visible items in kitchen preferences
- *tweak* stone-status-all: adds an option to toggle the economic status of all stones

Fixes

- Lua: registered `dfhack.constructions.designateRemove()` correctly
- *prospect*: fixed crash due to invalid vein materials
- *tweak* max-wheelbarrow: fixed conflict with building renaming
- *view-item-info*: stopped appending extra newlines permanently to descriptions

Misc Improvements

- Added logo to documentation
- Documented several missing `dfhack.gui` Lua functions
- *adv-rumors*: bound to Ctrl-A
- *command-prompt*: added support for `Gui::getSelectedPlant()`
- *gui/advfort*: bound to Ctrl-T
- *gui/room-list*: added support for `Gui::getSelectedBuilding()`
- *gui/unit-info-viewer*: bound to Alt-I

- *modtools/create-unit*: made functions available to other scripts
- ***search*:**
  - added support for stone restrictions screen (under `z`: Status)
  - added support for kitchen preferences (also under `z`)

API

- **New functions (all available to Lua as well):**
  - `Buildings::getRoomDescription()`
  - `Items::checkMandates()`
  - `Items::canTrade()`
  - `Items::canTradeWithContents()`
  - `Items::isRouteVehicle()`
  - `Items::isSquadEquipment()`
  - `Kitchen::addExclusion()`
  - `Kitchen::findExclusion()`
  - `Kitchen::removeExclusion()`
- syndrome-util: added `eraseSyndromeData()`

Internals

- Fixed compiler warnings on all supported build configurations
- Windows build scripts now work with non-C system drives

Structures

- `dfhack_room_quality_level`: new enum
- `glowing_barrier`: identified `triggered`, added comments
- `item_flags2`: renamed `has_written_content` to `unk_book`
- `kitchen_exc_type`: new enum (for `ui.kitchen`)
- `mandate.mode`: now an enum
- `unit_personality.emotions.flags.memory`: identified
- `viewscreen_kitchenprefst.forbidden`, `possible`: now a bitfield, `kitchen_pref_flag`
- `world_data.feature_map`: added extensive documentation (in XML)

9.4.24 DFHack 0.44.10-beta1

New Scripts

- *devel/find-primitive*: finds a primitive variable in memory

Fixes

- Units::getAnyUnit(): fixed a couple problematic conditions and potential segfaults if global addresses are missing
- *autodump*, *automelt*, *autotrade*, *stocks*, *stockpiles*: fixed conflict with building renaming
- *exterminate*: fixed documentation of `this` option
- ***full-heal*:**
  - units no longer have a tendency to melt after being healed
  - healed units are no longer treated as patients by hospital staff
  - healed units no longer attempt to clean themselves unsuccessfully
  - wounded fliers now regain the ability to fly upon being healing
  - now heals suffocation, numbness, infection, spilled guts and gelding
- ***modtools/create-unit*:**
  - creatures of the appropriate age are now spawned as babies or children where applicable
  - fix: civ_id is now properly assigned to historical_figure, resolving several hostility issues (spawned pets are no longer attacked by fortress military!)
  - fix: unnamed creatures are no longer spawned with a string of numbers as a first name
- *stockpiles*: stopped sidebar option from overlapping with *autodump*
- *tweak* block-labors: fixed two causes of crashes related in the v-p-l menu

Misc Improvements

- *blueprint*: added a basic Lua API
- *devel/export-dt-ini*: added tool offsets for DT 40
- *devel/save-version*: added current DF version to output
- *install-info*: added information on tweaks

Internals

- Added function names to DFHack's NullPointer and InvalidArgument exceptions
- Added `Gui::inRenameBuilding()`
- Linux: required plugins to have symbols resolved at link time, for consistency with other platforms

9.4.25 DFHack 0.44.10-alpha1

New Scripts

- *caravan*: adjusts properties of caravans
- *gui/autogems*: a configuration UI for the *autogems* plugin

Fixes

- Fixed uninitialized pointer being returned from `Gui::getAnyUnit()` in rare cases
- *autohauler*, *autolabor*, *labormanager*: fixed fencepost error and potential crash
- *dwarfvet*: fixed infinite loop if an animal is not accepted at a hospital
- *liquids*: fixed "range" command to default to 1 for dimensions consistently
- *search*: fixed 4/6 keys in unit screen search
- *view-item-info*: fixed an error with some armor

Misc Improvements

- *autogems*: can now blacklist arbitrary gem types (see *gui/autogems*)
- *exterminate*: added more words for current unit, removed warning
- *fpause*: now pauses worldgen as well

Internals

- Added some build scripts for Sublime Text
- Changed submodule URLs to relative URLs so that they can be cloned consistently over different protocols (e.g. SSH)

9.4.26 DFHack 0.44.09-r1

Fixes

- *modtools/item-trigger*: fixed token format in help text

Misc Improvements

- Reorganized changelogs and improved changelog editing process
- *modtools/item-trigger*: added support for multiple type/material/contaminant conditions

Internals

- OS X: Can now build with GCC 7 (or older)

Structures

- `army`: added vector new in 0.44.07
- `building_type`: added human-readable `name` attribute
- `furnace_type`: added human-readable `name` attribute
- `renderer`: fixed vtable addresses on 64-bit OS X
- `site_reputation_report`: named `reports` vector

- `workshop_type`: added human-readable `name` attribute

9.4.27 DFHack 0.44.09-alpha1

Fixes

- *digtype*: stopped designating non-vein tiles (open space, trees, etc.)
- *labormanager*: fixed crash due to dig jobs targeting some unrevealed map blocks

9.4.28 DFHack 0.44.08-alpha1

Fixes

- *fix/dead-units*: fixed a bug that could remove some arriving (not dead) units

9.4.29 DFHack 0.44.07-beta1

Misc Improvements

- *modtools/item-trigger*: added the ability to specify inventory mode(s) to trigger on

Structures

- Added symbols for Toady's 0.44.07 Linux test build to fix Bug 10615
- `world_site`: fixed alignment

9.4.30 DFHack 0.44.07-alpha1

Fixes

- Fixed some CMake warnings (CMP0022)
- Support for building on Ubuntu 18.04
- *embark-assistant*: fixed detection of reanimating biomes

Misc Improvements

- ***embark-assistant*:**
  - Added search for adamantine
  - Now supports saving/loading profiles
- *fillneeds*: added `-all` option to apply to all units
- *remotefortressreader*: added flows, instruments, tool names, campfires, ocean waves, spiderwebs

Structures

- Several new names in instrument raw structures
- `identity`: identified `profession`, `civ`
- `manager_order_template`: fixed last field type
- `viewscreen_createquotast`: fixed layout
- `world.language`: moved `colors`, `shapes`, `patterns` to `world.descriptors`
- **`world.reactions`, `world.reaction_categories`: moved to new compound, `world.reactions`. Requires rena**
  - `world.reactions` to `world.reactions.reactions`
  - `world.reaction_categories` to `world.reactions.reaction_categories`

9.4.31 DFHack 0.44.05-r2

New Plugins

- *embark-assistant*: adds more information and features to embark screen

New Scripts

- *adv-fix-sleepers*: fixes units in adventure mode who refuse to wake up (Bug 6798)
- *hermit*: blocks caravans, migrants, diplomats (for hermit challenge)

New Features

- With `PRINT_MODE:TEXT`, setting the `DFHACK_HEADLESS` environment variable will hide DF's display and allow the console to be used normally. (Note that this is intended for testing and is not very useful for actual gameplay.)

Fixes

- *devel/export-dt-ini*: fix language_name offsets for DT 39.2+
- *devel/inject-raws*: fixed gloves and shoes (old typo causing errors)
- *remotefortressreader*: fixed an issue with not all engravings being included
- *view-item-info*: fixed an error with some shields

Misc Improvements

- *adv-rumors*: added more keywords, including names
- *autochop*: can now exclude trees that produce fruit, food, or cookable items
- *remotefortressreader*: added plant type support

9.4.32 DFHack 0.44.05-r1

New Scripts

- *break-dance*: Breaks up a stuck dance activity
- *fillneeds*: Use with a unit selected to make them focused and unstressed
- *firestarter*: Lights things on fire: items, locations, entire inventories even!
- *flashstep*: Teleports adventurer to cursor
- *ghostly*: Turns an adventurer into a ghost or back
- *questport*: Sends your adventurer to the location of your quest log cursor
- *view-unit-reports*: opens the reports screen with combat reports for the selected unit

Fixes

- *devel/inject-raws*: now recognizes spaces in reaction names
- *dig*: added support for designation priorities - fixes issues with designations from `digv` and related commands having extremely high priority
- ***dwarfmonitor*:**
  - fixed display of creatures and poetic/music/dance forms on `prefs` screen
  - added "view unit" option
  - now exposes the selected unit to other tools
- *names*: fixed many errors
- *quicksave*: fixed an issue where the "Saving…" indicator often wouldn't appear

Misc Improvements

- *binpatch*: now reports errors for empty patch files
- *force*: now provides useful help
- ***full-heal*:**
  - can now select corpses to resurrect
  - now resets body part temperatures upon resurrection to prevent creatures from freezing/melting again
  - now resets units' vanish countdown to reverse effects of *exterminate*
- ***gui/gm-unit*:**
  - added a profession editor
  - misc. layout improvements
- *launch*: can now ride creatures
- *names*: can now edit names of units
- ***remotefortressreader*:**
  - support for moving adventurers

  - support for vehicles, gem shapes, item volume, art images, item improvements

Removed

- *tweak*: `kitchen-keys`: Bug 614 fixed in DF 0.44.04

Internals

- `Gui::getAnyUnit()` supports many more screens/menus

Structures

- New globals: `soul_next_id`

9.4.33 DFHack 0.44.05-alpha1

Misc Improvements

- *gui/liquids*: added more keybindings: 0-7 to change liquid level, P/B to cycle backwards

Structures

- `incident`: re-aligned again to match disassembly

9.4.34 DFHack 0.44.04-alpha1

Fixes

- *devel/inject-raws*: now recognizes spaces in reaction names
- *exportlegends*: fixed an error that could occur when exporting empty lists

Structures

- `artifact_record`: fixed layout (changed in 0.44.04)
- `incident`: fixed layout (changed in 0.44.01) - note that many fields have moved

9.4.35 DFHack 0.44.03-beta1

Fixes

- *autolabor*, *autohauler*, *labormanager*: added support for "put item on display" jobs and building/destroying display furniture
- *gui/gm-editor*: fixed an error when editing primitives in Lua tables

Misc Improvements

- *devel/dump-offsets*: now ignores `index` globals
- *gui/pathable*: added tile types to sidebar
- ***modtools/skill-change*:**
  - now updates skill levels appropriately
  - only prints output if `-loud` is passed

Structures

- Added `job_type.PutItemOnDisplay`
- Added `twbt_render_map` code offset on x64
- Fixed an issue preventing `enabler` from being allocated by DFHack
- Found `renderer` vtable on osx64
- **New globals:**
  - `version`
  - `min_load_version`
  - `movie_version`
  - `basic_seed`
  - `title`
  - `title_spaced`
  - `ui_building_resize_radius`
- `adventure_movement_optionst`, `adventure_movement_hold_tilest`, `adventure_movement_climbst`: named coordinate fields
- `mission`: added type
- `unit`: added 3 new vmethods: `getCreatureTile`, `getCorpseTile`, `getGlowTile`
- `viewscreen_assign_display_itemst`: fixed layout on x64 and identified many fields
- `viewscreen_reportliststst`: fixed layout, added `mission_id` vector
- `world.status`: named `missions` vector

9.4.36 DFHack 0.44.03-alpha1

Lua

- Improved `json` I/O error messages
- Stopped a crash when trying to create instances of classes whose vtable addresses are not available

9.4.37 DFHack 0.44.02-beta1

New Scripts

- *devel/check-other-ids*: Checks the validity of "other" vectors in the `world` global
- *gui/cp437-table*: An in-game CP437 table

Fixes

- Fixed issues with the console output color affecting the prompt on Windows
- *createitem*: stopped items from teleporting away in some forts
- *gui/gm-unit*: can now edit mining skill
- *gui/quickcmd*: stopped error from adding too many commands
- *modtools/create-unit*: fixed error when domesticating units

Misc Improvements

- The console now provides suggestions for built-in commands
- *devel/export-dt-ini*: avoid hardcoding flags
- ***exportlegends*:**
  - reordered some tags to match DF's order
  - added progress indicators for exporting long lists
- *gui/gm-editor*: added enum names to enum edit dialogs
- *gui/gm-unit*: made skill search case-insensitive
- *gui/rename*: added "clear" and "special characters" options
- ***remotefortressreader*:**
  - includes item stack sizes
  - some performance improvements

Removed

- *warn-stuck-trees*: Bug 9252 fixed in DF 0.44.01

Lua

- Exposed `get_vector()` (from C++) for all types that support `find()`, e.g. `df.unit.get_vector() == df.global.world.units.all`

Structures

- Added `buildings_other_id.DISPLAY_CASE`
- Fixed `unit` alignment
- Fixed `viewscreen_titlest.start_savegames` alignment
- Identified `historical_entity.unknown1b.deities` (deity IDs)
- Located `start_dwarf_count` offset for all builds except 64-bit Linux; *startdwarf* should work now

9.4.38 DFHack 0.44.02-alpha1

New Scripts

- *devel/dump-offsets*: prints an XML version of the global table included in in DF

Fixes

- Fixed a crash that could occur if a symbol table in symbols.xml had no content

Lua

- Added a new `dfhack.console` API
- API can now wrap functions with 12 or 13 parameters

Structures

- The former `announcements` global is now a field in `d_init`
- The `ui_menu_width` global is now a 2-byte array; the second item is the former `ui_area_map_width` global, which is now removed
- `world` fields formerly beginning with `job_` are now fields of `world.jobs`, e.g. `world.job_list` is now `world.jobs.list`

9.5 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

9.5.1 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 - e.g. 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 the destructor is not available, returns *false*. (This typically only occurs when trying to delete an instance of a DF class with virtual methods whose vtable address has not been found; it is impossible for `delete()` to determine the validity of `ref`.)

  > **Warning:** `ref` **must** be an object allocated with `new`, like in C++. Calling `obj.field:delete()` where `obj` was allocated with `new` will not work. After `delete()` returns, `ref` remains as a dangling pointer, like a raw C++ pointer would. Any accesses to `ref` after `ref:delete()` has been called are undefined behavior.

- `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 a `value` field of the right type. If the object's XML definition has a `ref-target` attribute, they will also have a read-only `ref_target` field set to the corresponding type object.

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.

9.5.2 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[,args...])`

  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 `ints`.

- `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.getDFHackBuildID()`
- `dfhack.getCompiledDFVersion()`
- `dfhack.getGitDescription()`
- `dfhack.getGitCommit()`
- `dfhack.getGitXmlCommit()`
- `dfhack.getGitXmlExpectedCommit()`
- `dfhack.gitXmlMatch()`
- `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.

Warning: When printing CP437-encoded text to the console (for example, names returned from `dfhack.TranslateName()`), use `print(dfhack.df2console(text))` to ensure proper display on all platforms.

- `dfhack.utf2df(string)`

  Convert a string from UTF-8 to DF's CP437 encoding.

- `dfhack.toSearchNormalized(string)`

  Replace non-ASCII alphabetic characters in a CP437-encoded string with their nearest ASCII equivalents, if possible, and returns a CP437-encoded string. Note that the returned string may be longer than the input string. For example, `ä` is replaced with `a`, and `æ` is replaced with `ae`.

- `dfhack.run_command(command[, ...])`

  Run an arbitrary DFHack command, with the core suspended, and send output to the DFHack console. The command can be passed as a table, multiple string arguments, or a single string argument (not recommended - in this case, the usual DFHack console tokenization is used).

  A `command_result` constant starting with `CR_` is returned, where `CR_OK` indicates success.

  The following examples are equivalent:

```
dfhack.run_command({'ls', '-a'})
dfhack.run_command('ls', '-a')
dfhack.run_command('ls -a')  -- not recommended
```

- `dfhack.run_command_silent(command[, ...])`

  Similar to `run_command()`, but instead of printing to the console, returns an `output, command_result` pair. `output` is a single string - see `dfhack.internal.runCommand()` to obtain colors as well.

Gui module

Screens

- `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.

General-purpose selections

- `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.getSelectedPlant([silent])`

  Returns the plant selected via `k`.

- `dfhack.gui.getAnyUnit(screen)`

- `dfhack.gui.getAnyItem(screen)`

- `dfhack.gui.getAnyBuilding(screen)`
- `dfhack.gui.getAnyPlant(screen)`

  Similar to the corresponding `getSelected` functions, but operate on the screen given instead of the current screen and always return `nil` silently on failure.

Fortress mode

- `dfhack.gui.getDwarfmodeViewDims()`

  Returns dimensions of the main fortress mode screen. See `getPanelLayout()` in the `gui.dwarfmode` module for a more Lua-friendly version.

- `dfhack.gui.resetDwarfmodeView([pause])`

  Resets the fortress mode sidebar menus and cursors to their default state. If `pause` is true, also pauses the game.

- `dfhack.gui.revealInDwarfmodeMap(pos)`

  Centers the view on the given position, which can be a `df.coord` instance or a table assignable to a `df.coord` (see *Recursive table assignment*), e.g.:

```
{x = 5, y = 7, z = 11}
getSelectedUnit().pos
copyall(df.global.cursor)
```

  Returns false if unsuccessful.

- `dfhack.gui.refreshSidebar()`

  Refreshes the fortress mode sidebar. This can be useful when making changes to the map, for example, because DF only updates the sidebar when the cursor position changes.

- `dfhack.gui.inRenameBuilding()`

  Returns `true` if a building is being renamed.

Announcements

- `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`.

Other

- `dfhack.gui.getDepthAt(x, y)`

  Returns the distance from the z-level of the tile at map coordinates (x, y) to the closest ground z-level below. Defaults to 0, unless overriden by plugins.

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,cooldown)`

  Prevent the worker from taking jobs at the specified workshop for the specified cooldown period (in ticks). This doesn't decrease the cooldown period in any circumstances.

- `dfhack.job.removeWorker(job,cooldown)`

  Removes the worker from the specified workshop job, and sets the cooldown period (using the same logic as `setJobCooldown`). 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, item_type)`

  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.getUnitsInBox(x1,y1,z1,x2,y2,z2[,filter])`

  Returns a table of all units within the specified coordinates. If the `filter` argument is given, only units where `filter(unit)` returns true will be included. Note that `pos2xyz()` cannot currently be used to convert coordinate objects to the arguments required by this function.

- `dfhack.units.teleport(unit, pos)`

  Moves the specified unit and any riders to the target coordinates, setting tile occupancy flags appropriately. Returns true if successful.

- `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.getOuterContainerRef(unit)`

  Returns a table (in the style of a `specific_ref` struct) of the outermost object that contains the unit (or one of the unit itself.) The `type` field contains a `specific_ref_type` of `UNIT`, `ITEM_GENERAL`, or `VERMIN_EVENT`. The `object` field contains a pointer to a unit, item, or vermin, respectively.

- `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.isActive(unit)`

  The unit is active (alive and on the map).

- `dfhack.units.isAlive(unit)`

  The unit isn't dead or undead.

- `dfhack.units.isDead(unit)`

  The unit is completely dead and passive, or a ghost. Equivalent to `dfhack.units.isKilled(unit) or dfhack.units.isGhost(unit)`.

- `dfhack.units.isKilled(unit)`

  The unit has been killed.

- `dfhack.units.isGhost(unit)`

  The unit is a ghost.

- `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.isFortControlled(unit)`

  Similar to `dfhack.units.isCitizen(unit)`, but is based on checks for units hidden in ambush, and includes tame animals. Returns *false* if not in fort mode.

- `dfhack.units.isVisible(unit)`

  The unit is visible on the map.

- `dfhack.units.isHidden(unit)`

  The unit is hidden to the player, accounting for sneaking. Works for any game mode.

- `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.isValidLabor(unit, unit_labor)`

  Returns whether the indicated labor is settable for the given unit.

- `dfhack.units.setLaborValidity(unit_labor, isValid)`

  Sets the given labor to the given (boolean) validity for all units that are part of your fortress civilization. Valid labors are allowed to be toggled in the in-game labor management screens (including DFHack's *labor manipulator screen*).

- `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.

- `dfhack.units.getGoalType(unit[,goalIndex])`

  Retrieves the goal type of the dream that the given unit has. By default the goal of the first dream is returned. The goalIndex parameter may be used to retrieve additional dream goals. Currently only one dream per unit is supported by Dwarf Fortress. Support for multiple dreams may be added in future versions of Dwarf Fortress.

- `dfhack.units.getGoalName(unit[,goalIndex])`

  Retrieves the short name describing the goal of the dream that the given unit has. By default the goal of the first dream is returned. The goalIndex parameter may be used to retrieve additional dream goals. Currently only one dream per unit is supported by Dwarf Fortress. Support for multiple dreams may be added in future versions of Dwarf Fortress.

- `dfhack.units.isGoalAchieved(unit[,goalIndex])`

  Checks if given unit has achieved the goal of the dream. By default the status of the goal of the first dream is returned. The goalIndex parameter may be used to check additional dream goals. Currently only one dream per unit is supported by Dwarf Fortress. Support for multiple dreams may be added in future versions of Dwarf Fortress.

- `dfhack.units.getStressCategory(unit)`

  Returns a number from 0-6 indicating stress. 0 is most stressed; 6 is least. Note that 0 is guaranteed to remain the most stressed but 6 could change in the future.

- `dfhack.units.getStressCategoryRaw(stress_level)`

  Identical to `getStressCategory` but takes a raw stress level instead of a unit.

- `dfhack.units.getStressCutoffs()`

  Returns a table of the cutoffs used by the above stress level functions.

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.getBookTitle(item)`

  Returns the title of the "book" item, or an empty string if the item isn't a "book" or it doesn't have a title. A "book" is a codex or a tool item that has page or writings improvements, such as scrolls and quires.

- `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.getOuterContainerRef(item)`

  Returns a table (in the style of a `specific_ref` struct) of the outermost object that contains the item (or one of the item itself.) The `type` field contains a `specific_ref_type` of `UNIT`, `ITEM_GENERAL`, or `VERMIN_EVENT`. The `object` field contains a pointer to a unit, item, or vermin, respectively.

- `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.

- `dfhack.items.createItem(item_type, item_subtype, mat_type, mat_index, unit)`

  Creates an item, similar to the *createitem* plugin.

- `dfhack.items.checkMandates(item)`

  Returns true if the item is free from mandates, or false if mandates prevent trading the item.

- `dfhack.items.canTrade(item)`

  Checks whether the item can be traded.

- `dfhack.items.canTradeWithContents(item)`

  Checks whether the item and all items it contains, if any, can be traded.

- `dfhack.items.isRouteVehicle(item)`

  Checks whether the item is an assigned hauling vehicle.

- `dfhack.items.isSquadEquipment(item)`

  Checks whether the item is assigned to a squad.

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.isTileVisible(coords)`, or `isTileVisible(x,y,z)`

  Checks if the given df::coord or x,y,z in local tile coordinates is visible.

- `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.getPlantAtTile(pos)`, or `getPlantAtTile(x,y,z)`

  Returns the plant struct that owns the tile at the specified position.

- `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,allow_wall])`

  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. Set `allow_wall` to true if the building is unhindered by walls (such as an activity zone).

- `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.

- `dfhack.buildings.getCageOccupants(cage)`

  Returns a list of units in the given built cage. Note that this is different from the list of units assigned to the cage, which can be accessed with `cage.assigned_units`.

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. You can specify a non-rectangular designation for building types that support extents by setting the `room.extents` bitmap before calling this function. The extents will be reset, however, if the size returned by this function doesn't match the input size parameter.

- `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.

- `dfhack.buildings.markedForRemoval(building)`

  Returns *true* if the building is marked for removal (with `x`), *false* otherwise.

- `dfhack.buildings.getRoomDescription(building[, unit])`

  If the building is a room, returns a description including quality modifiers, e.g. "Royal Bedroom". Otherwise, returns an empty string.

  The unit argument is passed through to DF and may modify the room's value depending on the unit given.

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`. If `room.extents` is assigned this way and this function returns with error, the memory allocated for the extents is freed.

- `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.

Kitchen module

- `dfhack.kitchen.findExclusion(type, item_type, item_subtype, mat_type, mat_index)`

  Finds a kitchen exclusion in the vectors in `df.global.ui.kitchen`. Returns -1 if not found.

  - `type` is a `df.kitchen_exc_type`, i.e. `df.kitchen_exc_type.Cook` or `df.kitchen_exc_type.Brew`.

  - `item_type` is a `df.item_type`
  - `item_subtype`, `mat_type`, and `mat_index` are all numeric

- `dfhack.kitchen.addExclusion(type, item_type, item_subtype, mat_type, mat_index)`
- `dfhack.kitchen.removeExclusion(type, item_type, item_subtype, mat_type, mat_index)`

  Adds or removes a kitchen exclusion, using the same parameters as `findExclusion`. Both return `true` on success and `false` on failure, e.g. when adding an exclusion that already exists or removing one that does not.

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

Common parameters to these functions include:

- `x`, `y`: screen coordinates in tiles; the upper left corner of the screen is `x = 0, y = 0`
- `pen`: a *pen object*
- `map`: a boolean indicating whether to draw to a separate map buffer (defaults to false, which is suitable for off-map text or a screen that hides the map entirely). Note that only third-party plugins like TWBT currently implement a separate map buffer. If no such plugins are enabled, passing `true` has no effect. However, this parameter should still be used to ensure that scripts work properly with such plugins.

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,map])`

  Paints a tile using given parameters. *See below* for a description of `pen`.

  Returns *false* on error, e.g. if coordinates are out of bounds

- `dfhack.screen.readTile(x,y[,map])`

  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[,map])`

  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[,map])`

  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.

Pen API

The `pen` argument used by `dfhack.screen` functions 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*.

Screen management

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:

- The *gui.Screen class* provides stubs for all of the functions listed below, and its use is recommended
- 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.restore_cwd()`

  Restores the current working directory to what it was when DF started.

- `dfhack.filesystem.get_initial_cwd()`

  Returns the value of the working directory when DF was started.

- `dfhack.filesystem.mkdir(path)`

  Creates a new directory. Returns `false` if unsuccessful, including if `path` already exists.

- `dfhack.filesystem.mkdir_recursive(path)`

  Creates a new directory, including any intermediate directories that don't exist yet. Returns `true` if the folder was created or already existed, or `false` if unsuccessful.

- `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[, include_prefix = true]])`

  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. Set `include_prefix` to false if you don't want the `path` string prepended to the returned filenames.

Console API

- `dfhack.console.clear()`

  Clears the console; equivalent to the `cls` built-in command.

- `dfhack.console.flush()`

  Flushes all output to the console. This can be useful when printing text that does not end in a newline but should still be displayed.

Internal API

These functions are intended for the use by dfhack developers, and are only documented here for completeness:

- `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)`

  Registers `path` as a *script path*. 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 list of *script 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.

- `dfhack.internal.runCommand(command[, use_console])`

  Runs a DFHack command with the core suspended. Used internally by the `dfhack.run_command()` family of functions.

  - `command`: either a table of strings or a single string which is parsed by the default console tokenization strategy (not recommended)
  - `use_console`: if true, output is sent directly to the DFHack console

  Returns a table with a `status` key set to a `command_result` constant (`status = CR_OK` indicates success). Additionally, if `use_console` is not true, enumerated table entries of the form `{color, text}` are included, e.g. `result[1][0]` is the color of the first piece of text printed (a `COLOR_` constant). These entries can be iterated over with `ipairs()`.

- `dfhack.internal.md5(string)`

  Returns the MD5 hash of the given string.

- `dfhack.internal.md5File(filename[,first_kb])`

  Computes the MD5 hash of the given file. Returns `hash, length` on success (where `length` is the number of bytes read from the file), or `nil, error` on failure.

  If the parameter `first_kb` is specified and evaluates to `true`, and the hash was computed successfully, a table containing the first 1024 bytes of the file is returned as the third return value.

- `dfhack.internal.threadid()`

  Returns a numeric identifier of the current thread.

Core interpreter context

While plugins can create any number of interpreter instances, there is one special context managed by the 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)`

  Creates a handler for state change events. Receives the same *SC_ 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`.

9.5.3 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
  ```

- State change event codes, used by `dfhack.onStateChange`

  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

- Command result constants (equivalent to `command_result` in C++), used by `dfhack.run_command()` and related functions:

  CR_OK, CR_LINK_FAILURE, CR_NEEDS_CONSOLE, CR_NOT_IMPLEMENTED, CR_FAILURE, CR_WRONG_USAGE, CR_NOT_FOUND

- 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.

- `printall_recurse(obj)`

  If the argument is a lua table or DF object reference, prints all fields recursively.

- `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.

- `ensure_key(t, key[, default_value])`

  If the Lua table `t` doesn't include the specified `key`, `t[key]` is set to the value of `default_value`, which defaults to `{}` if not set. The new or existing value of `t[key]` is then returned.

String class extentions

DFHack extends Lua's basic string class to include a number of convenience functions. These are invoked just like standard string functions, e.g.:

```
if imastring:startswith('imaprefix') then
```

- `string:startswith(prefix)`

  Returns `true` if the first `#prefix` characters of the string are equal to `prefix`. Note that `prefix` is not interpreted as a pattern.

- `string:endswith(suffix)`

  Returns `true` if the last `#suffix` characters of the string are equal to `suffix`. Note that `suffix` is not interpreted as a pattern.

- `string:split([delimiter[, plain]])`

  Split a string by the given delimiter. If no delimiter is specified, space (`' '`) is used. The delimter is treated as a pattern unless a `plain` is specified and set to `true`. To treat multiple successive delimiter characters as a single delimiter, e.g. to avoid getting empty string elements, pass a pattern like `' +'`. Be aware that passing patterns that match empty strings (like `' *'`) will result in improper string splits.

- `string:trim()`

  Removes spaces (i.e. everything that matches `'%s'`) from the start and end of a string. Spaces between non-space characters are left untouched.

- `string:wrap([width])`

  Inserts newlines into a string so no individual line exceeds the given width. Lines are split at space-separated word boundaries. Any existing newlines are kept in place. If a single word is longer than width, it is split over multiple lines. If width is not specified, 72 is used.

- `string:escape_pattern()`

  Escapes regex special chars in a string. E.g. `'a+b'` -> `'a%+b'`.

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.

argparse

The `argparse` module provides functions to help scripts process commandline parameters.

- `argparse.processArgs(args, validArgs)`

  A basic commandline processing function with simple syntax, useful if your script doesn't need the more advanced features of `argparse.processArgsGetopt()`.

  If `validArgs` is specified, it should contain a set of valid option names (without the leading dashes). For example:

```
argparse.processArgs(args, utils.invert{'opt1', 'opt2', 'opt3'})
```

  `processArgs` returns a map of option names it found in `args` to:

  - the token that came after the option

  - `''` if the next token was another option
  - a list of strings if the next token was `'['` (see below)

  Options in `args` from the commandline can be prefixed with either one dash (`'-'`) or two dashes (`'--'`). The user can add a backslash before the dash to allow a string to be identified as an option value instead of another option. For example: `yourscript --opt1 \-arg1`.

  If a `'['` token is found in `args`, the subsequent tokens will be interpreted as elements of a list until the matching closing `']'` is found. Brackets can be nested, but the inner brackets will be added to the list of tokens as literal `'['` and `']'` strings.

  Example commandlines:

```
yourscript --optName --opt2
yourscript --optName value
yourscript --optName [ list of values ]
yourscript --optName [ list of [ nested values ] [ in square brackets ] ]
yourscript --optName \--value
```

  Note that `processArgs` does not support non-option ("positional") parameters. They are supported by `processArgsGetopt` (see below).

- `argparse.processArgsGetopt(args, optionActions)`

  A fully-featured commandline processing function, with behavior based on the popular `getopt` library. You would use this instead of the simpler `processArgs` function if any of the following are true:

  - You want both short (e.g. `-f`) and aliased long-form (e.g. `--filename`) options
  - You have commandline components that are not arguments to options (e.g. you want to run your script like `yourscript command --verbose arg1 arg2 arg3` instead of `yourscript command --verbose --opt1 arg1 --opt2 arg2 --opt3 arg3`).
  - You want the convenience of combining options into shorter strings (e.g. `'-abcarg'` instead of `'-a -b -c arg'`)
  - You want to be able to parse and validate the option arguments as the commandline is being processed, as opposed to validating everything after commandline processing is complete.

  Commandlines processed by `processArgsGetopt` can have both "short" and "long" options, with each short option often having a long-form alias that behaves exactly the same as the short form. Short options have properties that make them very easy to type quickly by users who are familiar with your script options. Long options, on the other hand, are easily understandable by everyone and are useful in places where clarity is more important than brevity, e.g. in example commands. Each option can be configured to take an argument, which will be the string token that follows the option name on the commandline.

  Short options are a single letter long and are specified on a commandline by prefixing them with a single dash (e.g. the short option `a` would appear on the commandline as `-a`). Multiple successive short options that do not take arguments can be combined into a single option string (e.g. `'-abc'` instead of `'-a -b -c'`). Moreover, the argument for a short option can be appended directly to the single-letter option without an intervening space (e.g. `-d param` can be written as `-dparam`). These two convenience shorthand forms can be combined, allowing groups of short parameters to be written together, as long as at most the last short option takes an argument (e.g. combining the previous two examples into `-abcdparam`)

  Long options focus on clarity. They are usually entire words, or several words combined with hypens (`-`) or underscores (`_`). If they take an argument, the argument can be separated from the option name by a space or an equals sign (`=`). For example, the following two commandlines are equivalent: `yourscript --style pretty` and `yourscript --style=pretty`.

Another reason to use long options is if they represent an esoteric parameter that you don't expect to be commonly used and that you don't want to "waste" a single-letter option on. In this case, you can define a long option without a corresponding short option.

`processArgsGetopt` takes two parameters:

```
args: list of space-separated strings the user wrote on the commandline
optionActions: list of option specifications
```

and returns a list of positional parameters – that is, all strings that are neither options nor arguments to options. Options and positional parameters can appear in any order on the commandline, as long as arguments to options immediately follow the option itself.

Each option specification in `optionActions` has the following format: `{shortOptionName, longOptionAlias, hasArg=boolean, handler=fn}`

- `shortOptionName` is a one-character string (or `''` or `nil` if the parameter only has a long form). Numbers cannot be short options, and negative numbers (e.g. `'-10'`) will be interpreted as positional parameters and returned in the positional parameters list.
- `longOptionAlias` is an optional longer form of the short option name. If no short option name is specified, then this element is required.
- `hasArg` indicates whether the handler function for the option takes a parameter.
- `handler` is the handler function for the option. If `hasArg` is `true` then the next token on the commandline is passed to the handler function as an argument.

Example usage:

```
local args = {...}
local open_readonly, filename = false, nil      -- set defaults

local positionals = argparse.processArgsGetopt(args, {
  {'r', handler=function() open_readonly = true end},
  {'f', 'filename', hasArg=true,
   handler=function(optarg) filename = optarg end}
  })
```

In this example, if `args` is `{'first', '-rf', 'fname', 'second'}` or, equivalently, `{'first', '-r', '--filename', 'myfile.txt', 'second'}` (note the double dash in front of the long option alias), then `open_readonly` will be `true`, `filename` will be `'myfile.txt'` and `positionals` will be `{'first', 'second'}`.

- `argparse.stringList(arg, arg_name, list_length)`

  Parses a comma-separated sequence of strings and returns a lua list. Leading and trailing spaces are trimmed from the strings. If `arg_name` is specified, it is used to make error messages more useful. If `list_length` is specified and greater than `0`, then exactly that number of elements must be found or the function will error. Example:

```
stringList('hello , world,alist', 'words') => {'hello', 'world', 'alist'}
```

- `argparse.numberList(arg, arg_name, list_length)`

  Parses a comma-separated sequence of numeric strings and returns a list of the discovered numbers (as numbers, not strings). If `arg_name` is specified, it is used to make error messages more useful. If `list_length` is specified and greater than `0`, exactly that number of elements must be found or the function will error. Example:

```
numberList('10, -20 ,  30.5') => {10, -20, 30.5}
```

- `argparse.coords(arg, arg_name, skip_validation)`

  Parses a comma-separated coordinate string and returns a coordinate table of `{x, y, z}`. If the string `'here'` is passed, returns the coordinates of the active game cursor, or throws an error if the cursor is not active. This function also verifies that the coordinates are valid for the current map and throws if they are not (unless `skip_validation` is set to true).

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.

profiler

A third-party lua profiler module from http://lua-users.org/wiki/PepperfishProfiler. Module defines one function to create profiler objects which can be used to profile and generate report.

- `profiler.newProfiler([variant[, sampling_frequency]])`

  Returns an profile object with `variant` either `'time'` or `'call'`. `'time'` variant takes optional `sampling_frequency` parameter to select lua instruction counts between samples. Default is `'time'` variant with `10*1000` frequency.

  `'call'` variant has much higher runtime cost which will increase the runtime of profiled code by factor of ten. For the extreme costs it provides accurate function call counts that can help locate code which takes much time in native calls.

- `obj:start()`

  Resets collected statistics. Then it starts collecting new statistics.

- `obj:stop()`

  Stops profile collection.

- `obj:report(outfile[, sort_by_total_time])`

  Write a report from previous statistics collection to `outfile`. `outfile` should be writeable io file object (`io.open` or `io.stdout`). Passing `true` as second parameter `sort_by_total_time` switches sorting order to use total time instead of default self time order.

- `obj:prevent(function)`

  Adds an ignore filter for a `function`. It will ignore the pointed function and all of it children.

Examples

```
local prof = profiler.newProfiler()
prof:start()

profiledCode()

prof:stop()
```

(continues on next page)

(continued from previous page)

```
local out = io.open( "lua-profile.txt", "w+")
prof:report(out)
out:close()
```

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.

custom-raw-tokens

A module for reading custom tokens added to the raws by mods.

- `customRawTokens.getToken(typeDefinition, token)`

  Where `typeDefinition` is a type definition struct as seen in `df.global.world.raws` (e.g.: `dfhack.gui.getSelectedItem().subtype`) and `token` is the name of the custom token you want read. The arguments from the token will then be returned as strings using single or multiple return values. If the token is not present, the result is false; if it is present but has no arguments, the result is true. For `creature_raw`, it checks against no caste. For `plant_raw`, it checks against no growth.

- `customRawTokens.getToken(typeInstance, token)`

  Where `typeInstance` is a unit, entity, item, job, projectile, building, plant, or interaction instance. Gets `typeDefinition` and then returns the same as `getToken(typeDefinition, token)`. For units, it gets the token from the race or caste instead if appplicable. For plants growth items, it gets the token from the plant or plant growth instead if applicable. For plants it does the same but with growth number -1.

- `customRawTokens.getToken(raceDefinition, casteNumber, token)`

  The same as `getToken(unit, token)` but with a specified race and caste. Caste number -1 is no caste.

- `customRawTokens.getToken(raceDefinition, casteName, token)`

  The same as `getToken(unit, token)` but with a specified race and caste, using caste name (e.g. "FEMALE") instead of number.

- `customRawTokens.getToken(plantDefinition, growthNumber, token)`

  The same as `getToken(plantGrowthItem, token)` but with a specified plant and growth. Growth number -1 is no growth.

- `customRawTokens.getToken(plantDefinition, growthName, token)`

  The same as `getToken(plantGrowthItem, token)` but with a specified plant and growth, using growth name (e.g. "LEAVES") instead of number.

Examples:

- Using an eventful onReactionComplete hook, something for disturbing dwarven science:

```
if customRawTokens.getToken(reaction, "DFHACK_CAUSES_INSANITY") then
    -- make unit who performed reaction go insane
```

- Using an eventful onProjItemCheckMovement hook, a fast or slow-firing crossbow:

```
-- check projectile distance flown is zero, get firer, etc...
local multiplier = tonumber(customRawTokens.getToken(bow, "DFHACK_FIRE_RATE_
↪MULTIPLIER")) or 1
firer.counters.think_counter = firer.counters.think_counter * multiplier
```

- Something for a script that prints help text about different types of units:

```
local unit = dfhack.gui.getSelectedUnit()
if not unit then return end
local helpText = customRawTokens.getToken(unit, "DFHACK_HELP_TEXT")
if helpText then print(helpText) end
```

- Healing armour:

```
-- (per unit every tick)
local healAmount = 0
for _, entry in ipairs(unit.inventory) do
    if entry.mode == 2 then -- Worn
        healAmount = healAmount + tonumber((customRawTokens.getToken(entry.item,
↪"DFHACK_HEAL_AMOUNT")) or 0)
    end
end
unit.body.blood_count = math.min(unit.body.blood_max, unit.body.blood_count +␣
↪healAmount)
```

9.5.4 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. It also supports drawing to a separate map buffer if applicable (see `map()` below for details).

- `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 screen coordinates.

- `painter:cursorX()`

  Returns just the current *x* cursor coordinate

- `painter:cursorY()`

  Returns just the current *y* cursor coordinate

- `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:color(fg[,bold[,bg]])`

  Sets the specified colors of the current pen and returns *self*.

- `painter:key_pen(...)`

  Sets the current keybinding pen to `dfhack.pen.parse(old_pen,...)`, and returns *self*.

- `painter:map(to_map)`

  Enables or disables drawing to a separate map buffer. `to_map` is a boolean that will be passed as the `map` parameter to any `dfhack.screen` functions that accept it. Note that only third-party plugins like TWBT currently implement a separate map buffer; if none are enabled, this function has no effect (but should still be used to ensure proper support for such plugins). 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 `char()` 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*.

- `painter:key_string(keycode, text, ...)`

  A convenience wrapper around both `key()` and `string()` that prints both the specified keycode description and text, separated by `:`. Any extra arguments are passed directly to `string()`. Returns *self*.

Unless specified otherwise above, all Painter 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)`. Note that `onAboutToShow()` can dismiss active screens, and therefore change the potential parent. If parent is not specified, this function will re-detect the current topmost window after `self:onAboutToShow(parent)` returns.

- `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.

  Omitted fields are interpreted as having the value of 0.

- `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`.

- `autoarrange_subviews = bool` (default: false)
- `autoarrange_gap = int` (default: 0)

  If `autoarrange_subviews` is set to `true`, the Panel will automatically handle subview layout. Subviews are laid out vertically according to their current height, with `autoarrange_gap` empty lines between subviews. This allows you to have widgets dynamically change height or become visible/hidden and you don't have to worry about recalculating subview positions.

ResizingPanel class

Subclass of Panel; automatically adjusts its own frame height according to the size, position, and visibility of its subviews. Pairs nicely with a parent Panel that has `autoarrange_subviews` enabled.

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)`.

key If specified, the field is disabled until this key is pressed. Must be given as a string.

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.

text_hpen Specifies the pen for text hovered over by the mouse, if a click handler is registered.

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.

on_click A callback called when the label is clicked (optional)

on_rclick A callback called when the label is right-clicked (optional)

scroll_keys Specifies which keys the label should react to as a table. Default is `STANDARDSCROLL` (up or down arrows, page up or down).

show_scroll_icons Controls scroll icons' behaviour: `false` for no icons, `'right'` or `'left'` for icons next to the text in an additional column (`frame_inset` is adjusted to have `.r` or `.l` greater than `0`), `nil` same as `'right'` but changes `frame_inset` only if a scroll icon is actually necessary (if `getTextHeight()` is greater than `frame_body.height`). Default is `nil`.

up_arrow_icon The symbol for scroll up arrow. Default is `string.char(24)` (↑).

down_arrow_icon The symbol for scroll down arrow. Default is `string.char(25)` (↓).

scroll_icon_pen Specifies the pen for scroll icons. Default is `COLOR_LIGHTCYAN`.

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.

WrappedLabel class

This Label subclass represents text that you want to be able to dynamically wrap. This frees you from having to pre-split long strings into multiple lines in the Label `text` list.

It has the following attributes:

text_to_wrap The string (or a table of strings or a function that returns a string or a table of strings) to display. The text will be autowrapped to the width of the widget, though any existing newlines will be kept.

indent The number of spaces to indent the text from the left margin. The default is `0`.

The displayed text is refreshed and rewrapped whenever the widget bounds change. To force a refresh (to pick up changes in the string that `text_to_wrap` returns, for example), all `updateLayout()` on this widget or on a widget that contains this widget.

TooltipLabel class

This WrappedLabel subclass represents text that you want to be able to dynamically hide, like help text in a tooltip.

It has the following attributes:

show_tooltip Boolean or a callback; if true, the widget is visible.

The `text_pen` attribute of the `Label` class is overridden with a default of `COLOR_GREY` and the `indent` attribute of the `WrappedLabel` class is overridden with a default of `2`.

The text of the tooltip can be passed in the inherited `text_to_wrap` attribute so it can be autowrapped, or in the basic `text` attribute if no wrapping is required.

HotkeyLabel class

This Label subclass is a convenience class for formatting text that responds to a hotkey.

It has the following attributes:

key The hotkey keycode to display, e.g. `'CUSTOM_A'`.

label The string (or a function that returns a string) to display after the hotkey.

on_activate If specified, it is the callback that will be called whenever the hotkey is pressed.

CycleHotkeyLabel class

This Label subclass represents a group of related options that the user can cycle through by pressing a specified hotkey.

It has the following attributes:

key The hotkey keycode to display, e.g. `'CUSTOM_A'`.

label The string (or a function that returns a string) to display after the hotkey.

label_width The number of spaces to allocate to the `label` (for use in aligning a column of `CycleHotkeyLabel` labels).

options A list of strings or tables of `{label=string, value=string}`. String options use the same string for the label and value.

initial_option The value or numeric index of the initial option.

on_change The callback to call when the selected option changes. It is called as `on_change(new_option_value, old_option_value)`.

The index of the currently selected option in the `options` list is kept in the `option_idx` instance variable.

The CycleHotkeyLabel widget implements the following methods:

- `cyclehotkeylabel:cycle()`

  Cycles the selected option and triggers the `on_change` callback.

- `cyclehotkeylabel:getOptionLabel([option_idx])`

  Retrieves the option label at the given index, or the label of the currently selected option if no index is given.

- `cyclehotkeylabel:getOptionValue([option_idx])`

Retrieves the option value at the given index, or the value of the currently selected option if no index is given.

ToggleHotkeyLabel

This is a specialized subclass of CycleHotkeyLabel that has two options: `On` (with a value of `true`) and `Off` (with a value of `false`).

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.

edit_key If specified, the edit field is disabled until this key is pressed.

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. This is required for any entries where **text** is not a string.

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:getVisibleChoices()`

  Returns the *filtered* list of 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.

9.5.5 Plugins

DFHack plugins may export native functions and events to Lua contexts. These are exposed as `plugins.<name>` modules, which can be imported with `require('plugins.<name>')`. The plugins listed in this section expose functions and/or data to Lua in this way.

In addition to any native functions documented here, plugins that can be enabled (that is, plugins that support the *enable/disable API*) will have the following functions defined:

- `isEnabled()` returns whether the plugin is enabled.
- `setEnabled(boolean)` sets whether the plugin is enabled.

For plugin developers, note that a Lua file in `plugins/lua` is required for `require()` to work, even if it contains no pure-Lua functions. This file must contain `mkmodule('plugins.<name>')` to import any native functions defined in the plugin. See existing files in `plugins/lua` for examples.

blueprint

Lua functions provided by the *blueprint* plugin to programmatically generate blueprint files:

- `dig(start, end, name)`
- `build(start, end, name)`
- `place(start, end, name)`
- `query(start, end, name)`

  `start` and `end` are tables containing positions (see `xyz2pos`). `name` is used as the basis for the generated filenames.

The names of the functions are also available as the keys of the `valid_phases` table.

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:

a. tables of 4 numbers `{tile,fore,back,bright}` OR

b. empty table (tile not modified) OR

c. `{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
```

(continues on next page)

(continued from previous page)

```
    }
  }
```

Or with auto_gears:

```
require('plugins.building-hacks').registerBuilding{name="BONE_GRINDER",
  consume=15,
  auto_gears=true
  }
```

buildingplan

Native functions provided by the *buildingplan* plugin:

- `bool isPlannableBuilding(df::building_type type, int16_t subtype, int32_t custom)` returns whether the building type is handled by buildingplan.
- `bool isPlanModeEnabled(df::building_type type, int16_t subtype, int32_t custom)` returns whether the buildingplan UI is enabled for the specified building type.
- `bool isPlannedBuilding(df::building *bld)` returns whether the given building is managed by buildingplan.
- `void addPlannedBuilding(df::building *bld)` suspends the building jobs and adds the building to the monitor list.
- `void doCycle()` runs a check for whether buildlings in the monitor list can be assigned items and unsuspended. This method runs automatically twice a game day, so you only need to call it directly if you want buildingplan to do a check right now.
- `void scheduleCycle()` schedules a cycle to be run during the next non-paused game frame. Can be called multiple times while the game is paused and only one cycle will be scheduled.

burrows

The *burrows* plugin 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`.

cxxrandom

Exposes some features of the C++11 random number library to Lua.

Native functions (exported to Lua)

- `GenerateEngine(seed)`

  returns engine id

- `DestroyEngine(rngID)`

  destroys corresponding engine

- `NewSeed(rngID, seed)`

  re-seeds engine

- `rollInt(rngID, min, max)`

  generates random integer

- `rollDouble(rngID, min, max)`

  generates random double

- `rollNormal(rngID, avg, stddev)`
  generates random normal[gaus.]
- `rollBool(rngID, chance)`
  generates random boolean
- `MakeNumSequence(start, end)`
  returns sequence id
- `AddToSequence(seqID, num)`
  adds a number to the sequence
- `ShuffleSequence(seqID, rngID)`
  shuffles the number sequence
- `NextInSequence(seqID)`
  returns the next number in sequence

Lua plugin functions

- `MakeNewEngine(seed)`
  returns engine id

Lua plugin classes

`crng`

- `init(id, df, dist)`: constructor
  - `id`: Reference ID of engine to use in RNGenerations
  - `df` (optional): bool indicating whether to destroy the Engine when the crng object is garbage collected
  - `dist` (optional): lua number distribution to use
- `changeSeed(seed)`: alters engine's seed value
- `setNumDistrib(distrib)`: sets the number distribution crng object should use
  - `distrib`: number distribution object to use in RNGenerations
- `next()`: returns the next number in the distribution
- `shuffle()`: effectively shuffles the number distribution

`normal_distribution`

- `init(avg, stddev)`: constructor
- `next(id)`: returns next number in the distribution
  - `id`: engine ID to pass to native function

real_distribution

- `init(min, max)`: constructor
- `next(id)`: returns next number in the distribution
  - `id`: engine ID to pass to native function

int_distribution

- `init(min, max)`: constructor
- `next(id)`: returns next number in the distribution
  - `id`: engine ID to pass to native function

bool_distribution

- `init(chance)`: constructor
- `next(id)`: returns next boolean in the distribution
  - `id`: engine ID to pass to native function

num_sequence

- `init(a, b)`: constructor
- `add(num)`: adds num to the end of the number sequence
- `shuffle()`: shuffles the sequence of numbers
- `next()`: returns next number in the sequence

Usage

The basic idea is you create a number distribution which you generate random numbers along. The C++ relies on engines keeping state information to determine the next number along the distribution. You're welcome to try and (ab)use this knowledge for your RNG purposes.

Example:

```
local rng = require('plugins.cxxrandom')
local norm_dist = rng.normal_distribution(6820,116) // avg, stddev
local engID = rng.MakeNewEngine(0)
-- somewhat reminiscent of the C++ syntax
print(norm_dist:next(engID))

-- a bit more streamlined
local cleanup = true --delete engine on cleanup
local number_generator = rng.crng:new(engID, cleanup, norm_dist)
print(number_generator:next())

-- simplified
print(rng.rollNormal(engID,6820,116))
```

The number sequences are much simpler. They're intended for where you need to randomly generate an index, perhaps in a loop for an array. You technically don't need an engine to use it, if you don't mind never shuffling.

Example:

```
local rng = require('plugins.cxxrandom')
local g = rng.crng:new(rng.MakeNewEngine(0), true, rng.num_sequence:new(0,table_size))
g:shuffle()
for _ = 1, table_size do
    func(array[g:next()])
end
```

dig-now

The dig-now plugin exposes the following functions to Lua:

- **`dig_now_tile(pos)` or `dig_now_tile(x,y,z)`: Runs dig-now for the** specified tile coordinate. Default options apply, as if you were running the command `dig-now <pos> <pos>`. See the *dig-now* documentation for details on default settings.

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_items,call_native)`

   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")
```

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.

map-render

A way to ask DF to render a section of the fortress mode map. This uses a native DF rendering function so it's highly dependent on DF settings (e.g. tileset, colors, etc.)

Functions

- `render_map_rect(x,y,z,w,h)`

  returns a table with w*h*4 entries of rendered tiles. The format is same as `df.global.gps.screen` (tile,foreground,bright,background).

pathable

This plugin implements the back end of the *gui/pathable* script. It exports a single Lua function, in `hack/lua/plugins/pathable.lua`:

- `paintScreen(cursor[,skip_unrevealed])`: Paint each visible of the screen green or red, depending on whether it can be pathed to from the tile at `cursor`. If `skip_unrevealed` is specified and true, do not draw unrevealed tiles.

reveal

Native functions provided by the *reveal* plugin:

- `void unhideFlood(pos)`: Unhides map tiles according to visibility rules, starting from the given coordinates. This algorithm only processes adjacent hidden tiles, so it must start on a hidden tile in order to have any effect. It will not reveal hidden sections separated by already-unhidden tiles.

Example of revealing a cavern that happens to have an open tile at the specified coordinate:

```
unhideFlood({x=25, y=38, z=140})
```

sort

The *sort* plugin does not export any native functions as of now. Instead, it calls Lua code to perform the actual ordering of list items.

xlsxreader

Utility functions to facilitate reading .xlsx spreadsheets. It provides the following low-level API methods:

- `open_xlsx_file(filename)` returns a file_handle or nil on error
- `close_xlsx_file(file_handle)` closes the specified file_handle
- `list_sheets(file_handle)` returns a list of strings representing sheet names
- `open_sheet(file_handle, sheet_name)` returns a sheet_handle. This call always succeeds, even if the sheet doesn't exist. Non-existent sheets will have no data, though.
- `close_sheet(sheet_handle)` closes the specified sheet_handle
- `get_row(sheet_handle, max_tokens)` returns a list of strings representing the contents of the cells in the next row. The `max_tokens` parameter is optional. If set to a number > 0, it limits the number of cells read and returned for the row.

The plugin also provides Lua class wrappers for ease of use:

- `XlsxioReader` provides access to .xlsx files
- `XlsxioSheetReader` provides access to sheets within .xlsx files
- `open(filepath)` initializes and returns an `XlsxioReader` object

The `XlsxioReader` class has the following methods:

- `XlsxioReader:close()` closes the file. Be sure to close any open child sheet handles first!
- `XlsxioReader:list_sheets()` returns a list of strings representing sheet names
- `XlsxioReader:open_sheet(sheet_name)` returns an initialized `XlsxioSheetReader` object

The `XlsxioSheetReader` class has the following methods:

- `XlsxioSheetReader:close()` closes the sheet
- `XlsxioSheetReader:get_row(max_tokens)` reads the next row from the sheet. If `max_tokens` is specified and is a positive integer, only the first `max_tokens` elements of the row are returned.

Here is an end-to-end example:

```
local xlsxreader = require('plugins.xlsxreader')

local function dump_sheet(reader, sheet_name)
    print('reading sheet: ' .. sheet_name)
    local sheet_reader = reader:open_sheet(sheet_name)
    dfhack.with_finalize(
        function() sheet_reader:close() end,
        function()
            local row_cells = sheet_reader:get_row()
            while row_cells do
                printall(row_cells)
                row_cells = sheet_reader:get_row()
            end
        end
```

(continues on next page)

(continued from previous page)

```
    )
end

local filepath = 'path/to/some_file.xlsx'
local reader = xlsxreader.open(filepath)
dfhack.with_finalize(
    function() reader:close() end,
    function()
        for _,sheet_name in ipairs(reader:list_sheets()) do
            dump_sheet(reader, sheet_name)
        end
    end
)
```

9.5.6 Scripts

Any files with the `.lua` extension placed into the `hack/scripts` folder are automatically made avaiable as DFHack commands. The command corresponding to a script is simply the script's filename, relative to the scripts folder, with the extension omitted. For example:

- `hack/scripts/add-thought.lua` is invoked as `add-thought`
- `hack/scripts/gui/teleport.lua` is invoked as `gui/teleport`

Note: Scripts placed in subdirectories can be run as described above, but are not listed by the *ls* command unless `-a` is specified. In general, scripts should be placed in subfolders in the following situations:

- `devel`: scripts that are intended exclusively for DFHack development, including examples, or scripts that are experimental and unstable
- `fix`: fixes for specific DF issues
- `gui`: GUI front-ends for existing tools (for example, see the relationship between *teleport* and *gui/teleport*)
- `modtools`: scripts that are intended to be run exclusively as part of mods, not directly by end-users (as a rule of thumb: if someone other than a mod developer would want to run a script from the console, it should not be placed in this folder)

Scripts can also be placed in other folders - by default, these include `raw/scripts` and `data/save/`*`region`*`/raw/scripts`, but additional folders can be added (for example, a copy of the scripts repository for local development). See *Script paths* for more information on how to configure this behavior.

If the first line of the script is a one-line comment (starting with `--`), the content of the comment is used by the built-in `ls` and `help` commands. Such a comment is required for every script in the official DFHack repository.

Scripts are read from disk when run for the first time, or if they have changed since the last time they were run.

Each script has an isolated environment where global variables set by the script are stored. Values of globals persist across script runs in the same DF session. See *devel/lua-example* for an example of this behavior. Note that local variables do *not* persist.

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.

Additional data about how a script is invoked is passed to the script as a special `dfhack_flags` global, which is unique to each script. This table is guaranteed to exist, but individual entries may be present or absent depending on how the script was invoked. Flags that are present are described in the subsections below.

DFHack invokes the scripts in the *core context*; however it is possible to call them from any lua code (including from other scripts) in any context with `dfhack.run_script()` below.

General script API

- `dfhack.run_script(name[,args...])`

  Run a Lua script in hack/scripts/, as if it were started from the DFHack command-line. The `name` argument should be the name of the script without its extension, as it would be used on the command line.

  Example:

  In DFHack prompt:

```
repeat -time 14 -timeUnits days -command [ workorder ShearCreature ] -name␣
↪autoShearCreature
```

  In Lua script:

```
dfhack.run_script("repeat", "-time", "14", "-timeUnits", "days", "-command", "[",
↪"workorder", "ShearCreature", "]", "-name", "autoShearCreature")
```

  Note that the `dfhack.run_script()` function allows Lua errors to propagate to the caller.

  To run other types of commands (such as built-in commands, plugin commands, or Ruby scripts), see `dfhack.run_command()`. Note that this is slightly slower than `dfhack.run_script()` for Lua scripts.

- `dfhack.script_help([name, [extension]])`

  Returns the contents of the embedded documentation of the specified script. `extension` defaults to "lua", and `name` defaults to the name of the script where this function was called. For example, the following can be used to print the current script's help text:

```
local args = {...}
if args[1] == 'help' then
    print(script_help())
    return
end
```

Importing scripts

- `dfhack.reqscript(name)` or `reqscript(name)`

  Loads a Lua script and returns its environment (i.e. a table of all global functions and variables). This is similar to the built-in `require()`, but searches all script paths for the first matching `name.lua` file instead of searching the Lua library paths (like `hack/lua`).

Most scripts can be made to support `reqscript()` without significant changes (in contrast, `require()` requires the use of `mkmodule()` and some additional boilerplate). However, because scripts can have side effects when they are loaded (such as printing messages or modifying the game state), scripts that intend to support being imported must satisfy some criteria to ensure that they can be imported safely:

1. Include the following line - `reqscript()` will fail if this line is not present:

```
--@ module = true
```

2. Include a check for `dfhack_flags.module`, and avoid running any code that has side-effects if this flag is true. For instance:

```
-- (function definitions)
if dfhack_flags.module then
    return
end
-- (main script code with side-effects)
```

or:

```
-- (function definitions)
function main()
    -- (main script code with side-effects)
end
if not dfhack_flags.module then
    main()
end
```

Example usage:

```
local addThought = reqscript('add-thought')
addThought.addEmotionToUnit(unit, ...)
```

Circular dependencies between scripts are supported, as long as the scripts have no side-effects at load time (which should already be the case per the above criteria).

Warning: Avoid caching the table returned by `reqscript()` beyond storing it in a local or global variable as in the example above. `reqscript()` is fast for scripts that have previously been loaded and haven't changed. If you retain a reference to a table returned by an old `reqscript()` call, this may lead to unintended behavior if the location of the script changes (e.g. if a save is loaded or unloaded, or if a *script path* is added in some other way).

Tip

Mods that include custom Lua modules can write these modules to support `reqscript()` and distribute them as scripts in `raw/scripts`. Since the entire `raw` folder is copied into new saves, this will allow saves to be successfully transferred to other users who do not have the mod installed (as long as they have DFHack installed).

Backwards compatibility notes

For backwards compatibility, `moduleMode` is also defined if `dfhack_flags.module` is defined, and is set to the same value. Support for this may be removed in a future version.

- `dfhack.script_environment(name)`

  Similar to `reqscript()` but does not enforce the check for module support. This can be used to import scripts that support being used as a module but do not declare support as described above, although it is preferred to update such scripts so that `reqscript()` can be used instead.

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, the `dfhack_flags` table passed to the script will have 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

Example usage:

```
--@ enable = true
-- (function definitions...)
if dfhack_flags.enable then
    if dfhack_flags.enable_state then
        start()
    else
        stop()
    end
end
```

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`.

9.6 DFHack Remote Interface

DFHack provides a remote access interface that external tools can connect to and use to interact with DF. This is implemented with Google protobuf messages exchanged over a TCP socket. Both the core and plugins can define remotely-accessible methods, or **RPC methods**. The RPC methods currently available are not comprehensive, but can be extended with plugins.

Contents

9.6.1 Server configuration

DFHack attempts to start a TCP server to listen for remote connections on startup. If this fails (due to the port being in use, for example), an error message will be logged to stderr.log.

The server can be configured by setting options in `dfhack-config/remote-server.json`:

- `allow_remote` (default: `false`): if true, allows connections from hosts other than the local machine. This is insecure and may allow arbitrary code execution on your machine, so it is disabled by default.
- `port` (default: `5000`): the port that the remote server listens on. Overriding this will allow the server to work if you have multiple instances of DF running, or if you have something else running on port 5000. Note that the `DFHACK_PORT` *environment variable* takes precedence over this setting and may be more useful for overriding the port temporarily.

9.6.2 Developing with the remote API

At a high level, the core and plugins define RPC methods, and external clients can call these methods. Each method is assigned an ID internally, which clients use to call it. These method IDs can be obtained using the special `BindMethod` method, which has an ID of 0.

Examples

The *dfhack-run* command uses the RPC interface to invoke DFHack commands (or Lua functions) externally.

Plugins that implement RPC methods include:

- *rename*
- *remotefortressreader*
- *isoworldremote*

Plugins that use the RPC API include:

- stonesense

Third-party tools that use the RPC API include:

- Armok Vision (Bay12 forums thread)

Client libraries

Some external libraries are available for interacting with the remote interface from other (non-C++) languages, including:

- RemoteClientDF-Net for C#
- dfhackrpc for Go
- dfhack-remote for JavaScript
- dfhack-client-qt for C++ with Qt
- dfhack-client-python for Python (adapted from "Blendwarf")
- dfhack-client-java for Java
- dfhack-remote for Rust

9.6.3 Protocol description

This is a low-level description of the RPC protocol, which may be useful when developing custom clients.

A WireShark dissector for this protocol is available in the df_misc repo.

Built-in messages

These messages have hardcoded IDs; all others must be obtained through `BindMethod`.

ID	Method	Input	Output
0	BindMethod	dfproto.CoreBindRequest	dfproto.CoreBindReply
1	RunCommand	dfproto.CoreRunCommandRequest	dfproto.EmptyMessage

Conversation flow

- Client → Server: *handshake request*
- Server → Client: *handshake reply*
- **Repeated 0 or more times:**
  - Client → Server: *request*
  - Server → Client: *text* (0 or more times)

  - Server → Client: *result* or *failure*
- Client → Server: *quit*

Raw message types

- All numbers are little-endian
- All strings are ASCII
- A payload size of greater than 64MiB is an error
- See `RemoteClient.h` for definitions of constants starting with `RPC`

handshake request

Type	Name	Value
char[8]	magic	`DFHack?\n`
int32_t	version	1

handshake reply

Type	Name	Value
char[8]	magic	`DFHack!\n`
int32_t	version	1

header

Note: the two fields of this message are sometimes repurposed. Uses of this message are represented as `header(x, y)`, where `x` corresponds to the `id` field and `y` corresponds to `size`.

Type	Name
int16_t	id
int16_t	(padding - unused)
int32_t	size

request

Type	Description
header	`header(id, size)`
buffer	Protobuf-encoded payload of the input message type of the method specified by `id`; length of `size` bytes

text

Type	Description
header	`header(RPC_REPLY_TEXT, size)`
buffer	Protobuf-encoded payload of type `dfproto.CoreTextNotification`; length of `size` bytes

result

Type	Description
header	`header(RPC_REPLY_RESULT, size)`
buffer	Protobuf-encoded payload of the output message type of the oldest incomplete method call; when received, that method call is considered completed. Length of `size` bytes.

failure

Type	Description
header	`header(RPC_REPLY_FAIL, command_result)`
command_result	return code of the command (a constant starting with `CR_`; see `RemoteClient.h`)

quit

Note: the server closes the connection after receiving this message.

Type	Description
header	`header(RPC_REQUEST_QUIT, 0)`

9.7 DFHack API Reference

9.7.1 Maps API

DFHack offers several ways to access and manipulate map data.

- C++: the `Maps` and `MapCache` modules
- Lua: the dfhack.maps module
- All languages: the `map` field of the `world` global contains raw map data when the world is loaded.

Note: This page will eventually go into more detail about the available APIs. For now, it is just an overview of how DF map data is structured.

Contents

Tiles

The DF map has several types of tiles:

- **Local tiles** are at the smallest scale. In regular fortress/adventure mode play, the cursor takes up 1 local tile.

  Objects that use local tile coordinates include:

  - Units
  - Items
  - Projectiles

- **Blocks** are 16 × 16 × 1 groups of local tiles. Internally, many tile details are stored at the block level for space-efficiency reasons. Blocks are visible during zoomed-in fast travel in adventure mode.

  Objects that use block coordinates include:

  - Armies

- **Region tiles** are 3 × 3 groups of columns of blocks (they span the entire z-axis), or 48 × 48 columns of local tiles. DF sometimes refers to these as "mid-level tiles" (MLTs). Region tiles are visible when resizing a fortress before embarking, or in zoomed-out fast travel in adventure mode.

- **World tiles** are

  - 16 × 16 groups of region tiles, or
  - 48 × 48 groups of columns of blocks, or
  - 768 × 768 groups of columns of local tiles

  World tiles are visible on the world map before embarking, as well as in the civilization map in fortress mode and the quest log in adventure mode.

- Some map features are stored in 16 × 16 groups of world tiles, sometimes referred to as "feature shells".

9.8 DFHack Documentation System

DFHack documentation, like the file you are reading now, is created as `.rst` files, which are in reStructuredText (reST) format. This is a documentation format common in 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 is disabled by default due to the additional Python and Sphinx requirements. You typically only need to build the docs if you're changing them, or perhaps if you want a local HTML copy; otherwise, you can read an online version hosted by ReadTheDocs.

(Note that even if you do want a local copy, it is certainly not necessary 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.)

Contents

9.8.1 Documentation standards

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.

Command documentation

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 - anything between the tokens is copied into the appropriate scripts documentation page. For Ruby, we follow the built-in docstring convention (`=begin` and `=end`). For Lua, the tokens are `[====[` and `]====]` - ordinary multi-line strings. It is highly encouraged to reuse this string as the in-console documentation by (e.g.) printing it when a `-help` argument is given.

The docs **must** have a heading which exactly matches the command, underlined with `=====` to the same length. For example, a lua file would have:

```
local helpstr = [====[

add-thought
```

(continues on next page)

(continued from previous page)

```
==========
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.

]====]
```

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 in the terminal - Sphinx (our documentation system) will make sure paragraphs flow.

Command usage

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.
```

Links

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: e.g. `` `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 have link targets created automatically.

Note that the DFHack documentation is configured so that single backticks (with no prefix or suffix) produce links to internal link targets, such as the `autolabor` target shown above. This is different from the reStructuredText default behavior of rendering such text in italics (as a reference to a title). For alternative link behaviors, see:

- The reStructuredText documentation on roles

- The reStructuredText documentation on external links
- The Sphinx documentation on roles
  - `:doc:` is useful for linking to another document

9.8.2 Required dependencies

In order to build the documentation, you must have Python with Sphinx version 1.8 or later. Python 3 is recommended.

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.

Once you have installed Sphinx, `sphinx-build --version` should report the version of Sphinx that you have installed. If this works, CMake should also be able to find Sphinx.

For more detailed platform-specific instructions, see the sections below:

Linux

Most Linux distributions will include Python by default. If not, start by installing Python (preferably Python 3). On Debian-based distros:

```
sudo apt install python3
```

Check your package manager to see if Sphinx 1.8 or later is available. On Debian-based distros, this package is named `python3-sphinx`. If this package is new enough, you can install it directly. If not, or if you want to use a newer Sphinx version (which may result in faster builds), you can install Sphinx through the `pip` package manager instead. On Debian-based distros, you can install pip with:

```
sudo apt install python3-pip
```

Once pip is available, you can then install Sphinx with:

```
pip3 install sphinx
```

If you run this as an unprivileged user, it may install a local copy of Sphinx for your user only. The `sphinx-build` executable will typically end up in `~/.local/bin/` in this case. Alternatively, you can install Sphinx system-wide by running pip with `sudo`. In any case, you will need the folder containing `sphinx-build` to be in your `$PATH`.

macOS

macOS 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 directly from Homebrew:

```
brew install sphinx-doc
```

This will install Sphinx for macOS'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 macOS's system Python 2.7 is fine.

Windows

Python for Windows can be downloaded from python.org. The latest version of Python 3 is recommended, as it includes pip already.

You can also install Python and pip through the Chocolatey package manager. After installing Chocolatey as outlined in the *Windows compilation instructions*, run the following command from an elevated (admin) command prompt (e.g. `cmd.exe`):

```
choco install python pip -y
```

Once you have pip available, you can install Sphinx with the following command:

```
pip install sphinx
```

Note that this may require opening a new (admin) command prompt if you just installed pip from the same command prompt.

9.8.3 Building the documentation

Once the required dependencies are installed, there are multiple ways to run Sphinx to build the docs:

Using CMake

Enabling the `BUILD_DOCS` CMake option will cause the documentation to be built whenever it changes as part of the normal DFHack build process. There are several ways to do this:

- When initially running CMake, add `-DBUILD_DOCS:bool=ON` to your `cmake` command. For example:

```
cmake .. -DCMAKE_BUILD_TYPE:string=Release -DBUILD_DOCS:bool=ON -DCMAKE_INSTALL_
↪PREFIX=<path to DF>
```

- If you have already run CMake, you can simply run it again from your build folder to update your configuration:

```
cmake .. -DBUILD_DOCS:bool=ON
```

- You can edit the `BUILD_DOCS` setting in CMakeCache.txt directly
- You can use the CMake GUI or `ccmake` to change the `BUILD_DOCS` setting
- On Windows, if you prefer to use the batch scripts, you can run `generate-msvc-gui.bat` and set `BUILD_DOCS` through the GUI. If you are running another file, such as `generate-msvc-all.bat`, you will need to edit it to add the flag. You can also run `cmake` on the command line, similar to other platforms.

The generated documentation will be stored in `docs/html` in the root DFHack folder, and will be installed to `hack/docs` when you next install DFHack in a DF folder.

Running Sphinx manually

You can also build the documentation without running CMake - this is faster if you only want to rebuild the documentation regardless of any code changes. There is a `docs/build.sh` script provided for Linux and macOS that will run essentially the same command that CMake runs when building the docs - see the script for additional options.

To build the documentation with default options, run the following command from the root DFHack folder:

```
sphinx-build . docs/html
```

The resulting documentation will be stored in `docs/html` (you can specify a different path when running `sphinx-build` manually, but be warned that Sphinx may overwrite existing files in this folder).

Sphinx has many options to enable clean builds, parallel builds, logging, and more - run `sphinx-build --help` for details.

Building a PDF version

ReadTheDocs automatically builds a PDF version of the documentation (available under the "Downloads" section when clicking on the release selector). If you want to build a PDF version locally, you will need `pdflatex`, which is part of a TeX distribution. The following command will then build a PDF, located in `docs/pdf/latex/DFHack.pdf`, with default options:

```
sphinx-build -M latexpdf . docs/pdf
```

There is a `docs/build-pdf.sh` script provided for Linux and macOS that runs this command for convenience - see the script for additional options.

9.8.4 Building the changelogs

If you have Python installed, you can build just the changelogs without building the rest of the documentation by running the `docs/gen_changelog.py` script. This script provides additional options, including one to build individual changelogs for all DFHack versions - run `python docs/gen_changelog.py --help` for details.

Changelog entries are obtained from `changelog.txt` files in multiple repos. This allows changes to be listed in the same repo where they were made. These changelogs are combined as part of the changelog build process:

- `docs/changelog.txt` for changes in the main `dfhack` repo
- `scripts/changelog.txt` for changes made to scripts in the `scripts` repo
- `library/xml/changelog.txt` for changes made in the `df-structures` repo

Building the changelogs generates two files: `docs/_auto/news.rst` and `docs/_auto/news-dev.rst`. These correspond to *Changelog* and *Development Changelog* and contain changes organized by stable and development DFHack releases, respectively. For example, an entry listed under "0.44.05-alpha1" in changelog.txt will be listed under that version in the development changelog as well, but under "0.44.05-r1" in the stable changelog (assuming that is the closest stable release after 0.44.05-alpha1). An entry listed under a stable release like "0.44.05-r1" in changelog.txt will be listed under that release in both the stable changelog and the development changelog.

Changelog syntax

changelog.txt uses a syntax similar to RST, with a few special sequences:

- === indicates the start of a comment
- # indicates the start of a release name (do not include "DFHack")

- `##` indicates the start of a section name (this must be listed in `gen_changelog.py`)
- `-` indicates the start of a changelog entry. **Note:** an entry currently must be only one line.
- **`:` (colon followed by space) separates the name of a feature from a description of a change to that feature.** Changes made to the same feature are grouped if they end up in the same section.
- `:\` (colon, backslash, space) avoids the above behavior
- **`- @` (the space is optional) indicates the start of an entry that should only be displayed in NEWS-dev.rst.** Use this sparingly, e.g. for immediate fixes to one development build in another development build that are not of interest to users of stable builds only.
- Three `[` characters indicate the start of a block (possibly a comment) that spans multiple lines. Three `]` characters indicate the end of such a block.
- `!` immediately before a phrase set up to be replaced (see gen_changelog.py) stops that occurrence from being replaced.

9.8.5 GitHub Actions

Documentation is built automatically with GitHub Actions (a GitHub-provided continuous integration service) for all pull requests and commits in the "dfhack" and "scripts" repositories. These builds run with strict settings, i.e. warnings are treated as errors. If a build fails, you will see a red "x" next to the relevant commit or pull request. You can view detailed output from Sphinx in a few ways:

- Click on the red "x" (or green checkmark), then click "Details" next to the "Build / docs" entry
- For pull requests only: navigate to the "Checks" tab, then click on "Build" in the sidebar to expand it, then "docs" under it

Sphinx output will be visible under the step named "Build docs". If a different step failed, or you aren't sure how to interpret the output, leave a comment on the pull request (or commit).

You can also download the "docs" artifact from the summary page (typically accessible by clicking "Build") if the build succeeded. This is a way to visually inspect what the documentation looks like when built without installing Sphinx locally, although we recommend installing Sphinx if you are planning to do any significant work on the documentation.

9.9 DF data definitions (DF-structures)

DFHack's information about DF's data structures is stored in XML files in the df-structures repository. If you have *obtained a local copy of the DFHack source*, DF-structures is included as a submodule in `library/xml`.

Data structure layouts are described in files named with the `df.*.xml` pattern. This information is transformed by a Perl script (`codegen.pl`) into C++ headers, as well as metadata for the Lua wrapper. This ultimately allows DFHack code to access DF data directly, with the same speed and capabilities as DF itself, which is an advantage over the older out-of-process approach (used by debuggers and utilities like Dwarf Therapist). The main disadvantage of this approach is that any compiled code relying on these layouts will break when DF's layout changes, and will need to be recompiled for every new DF version.

Addresses of DF global objects and vtables are stored in a separate file, `symbols.xml`. Since these are only absolute addresses, they do not need to be compiled in to DFHack code, and are instead loaded at runtime. This makes fixes and additions to global addresses possible without recompiling DFHack. In an installed copy of DFHack, this file can be found at the root of the `hack` folder.

The following pages contain more detailed information about various aspects of DF-structures:

9.9.1 Data Structure Definition Syntax

Contents

This document documents the XML syntax used to define DF data structures for use in dfhack.

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.

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.

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.

Tagged unions

Union compounds and vectors of union compounds can additionally have `union-tag-field` and `union-tag-attr` attributes.

`union-tag-field` sets the name of the field that holds the tag for the union. Union compounds must have tags that are enumeration fields, while vectors of union compounds can have tags that are vectors of an enumeration type, or in the case of a union with exactly 2 members, a bit vector.

`union-tag-attr` overrides the name used to find the union member. By default, the field with a name equal to the enum key is chosen. When this attribute is set, the specified enum attr will be used instead.

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.

`<df-linked-list-type type-name='foo_link' item-type='foo'/>`

Defines a DF-style linked list node. This translates to:

```
<struct-type type-name='foo_link'>
    <pointer name='item' type-name='foo'/>
    <pointer name='prev' type-name='foo_link'/>
    <pointer name='next' type-name='foo_link'/>
</struct-type>
```

with some extra code to make it easier to interact with.

`<df-other-vectors-type type-name='foo_other' index-enum='foo_other_id' item-type='foo'/>`

Defines a tuple of vectors with the same base type. Individual vectors act as if they were defined as:

```
<stl-vector name='FOO_KEY' pointer-type='foo'/>
```

where `FOO_KEY` is a key in the `foo_other_id` enum.

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.

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
```

(continues on next page)

(continued from previous page)

```
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.

9.9.2 Updating DF-structures for a new DF version

Contents

General Process

Download the new versions. The scripts expect the following directory layout:

```
~userhome/
    Games/
        DF/
            df_linux/    - Current DF for linux
```

(continues on next page)

(continued from previous page)

```
            df_windows/  - Current DF for windows
            df_osx/      - Current DF for osx

            metasm/      - Checkout of the library for ruby scripts
                           from https://github.com/jjyg/metasm
            df_misc/     - Checkout of the ruby scripts
                           from https://github.com/jjyg/df_misc
            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.

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'
```

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

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.

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.

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 . . .

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.

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.

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 ...

9.10 Memory research

There are a variety of tools that can be used to analyze DF memory - some are listed here. Note that some of these may be old and unmaintained. If you aren't sure what tool would be best for your purposes, feel free to ask for advice (on IRC, Bay12, etc.).

Contents

9.10.1 Cross-platform tools

Ghidra

Ghidra is a cross-platform reverse-engineering framework (written in Java) available at https://ghidra-sre.org. It supports analyzing both 32-bit and 64-bit executables for all supported DF platforms. There are some custom DFHack Ghidra scripts available in the df_misc repo (look for `.java` files).

IDA Freeware 7.0

Available from Hex-Rays. Supports analyzing both 32-bit and 64-bit executables for all supported DF platforms. Some `.idc` scripts for IDA are available in the df_misc repo.

Hopper

Runs on macOS and some Linux distributions; available from https://www.hopperapp.com/. TWBT uses this to produce some patches.

DFHack tools

Plugins

There are a few development plugins useful for low-level memory research. They are not built by default, but can be built by setting the `BUILD_DEVEL` *CMake option*. These include:

- `check-structures-sanity`, which performs sanity checks on the given DF object. Note that this will crash in several cases, some intentional, so using this with *GDB* is recommended.
- `memview`, which produces a hex dump of a given memory range. It also highlights valid pointers, and can be configured to work with *Sizecheck* to auto-detect object sizes.
- `vectors`, which can identify instances of `std::vector` in a given memory range.

Scripts

Several *development scripts* can be useful for memory research. These include (but are not limited to):

- *devel/dump-offsets*
- *devel/find-offsets*
- *devel/lsmem*
- *devel/sc* (requires *Sizecheck*)
- *devel/visualize-structure*
- Generally, any script starting with `devel/find`

Sizecheck

Sizecheck is a custom tool that hooks into the memory allocator and inserts a header indicating the size of every object. The corresponding logic to check for this header when freeing memory usually works, but is inherently not foolproof. You should not count on DF being stable when using this.

DFHack's implementation of sizecheck is currently only tested on Linux, although it probably also works on macOS. It can be built with the `BUILD_SIZECHECK` *CMake option*, which produces a `libsizecheck` library installed in the `hack` folder. On Linux, passing `--sc` as the first argument to the `dfhack` launcher script will load this library on startup. On other platforms, or when passing a different argument to the launcher (such as for *GDB*), you will need to preload this library manually, by setting `PRELOAD_LIB` on Linux (or `LD_PRELOAD` if editing the `dfhack` launcher script directly), or by editing the `dfhack` launcher script and adding the library to `DYLD_INSERT_LIBRARIES` on macOS.

There is also an older sizecheck implementation by Mifki available on GitHub (`b.cpp` is the main sizecheck library, and `win_patch.cpp` is used for Windows support). To use this with other DFHack tools, you will likely need to edit the header's magic number to match what is used in *devel/sc* (search for a hexadecimal constant starting with `0x`).

Legacy tools

Some very old DFHack tools are available in the legacy branch on GitHub. No attempt is made to support these.

9.10.2 Linux-specific tools

GDB

GDB is technically cross-platform, but tends to work best on Linux, and DFHack currently only offers support for using GDB on 64-bit Linux. To start with GDB, pass `-g` to the DFHack launcher script:

```
./dfhack -g
```

Some basic GDB commands:

- `run`: starts DF from the GDB prompt. Any arguments will be passed as command-line arguments to DF (e.g. *load-save* may be useful).
- `bt` will produce a backtrace if DF crashes.

See the official GDB documentation for more details.

Other analysis tools

The `dfhack` launcher script on Linux has support for launching several other tools alongside DFHack, including Valgrind (as well as Callgrind and Helgrind) and strace. See the script for the exact command-line option to specify. Note that currently only one tool at a time is supported, and must be specified with the first argument to the script.

df-structures GUI

This is a tool written by Angavrilov and available on GitHub. It only supports 32-bit DF. Some assistance may be available on IRC.

EDB (Evan's debugger)

Available on GitHub.

9.10.3 Windows-specific tools

Some people have used Cheat Engine for research in the past.

9.11 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.

Contents

9.11.1 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 the patches folder in commit b0e1b51 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.

9.11.2 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.

9.11.3 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.

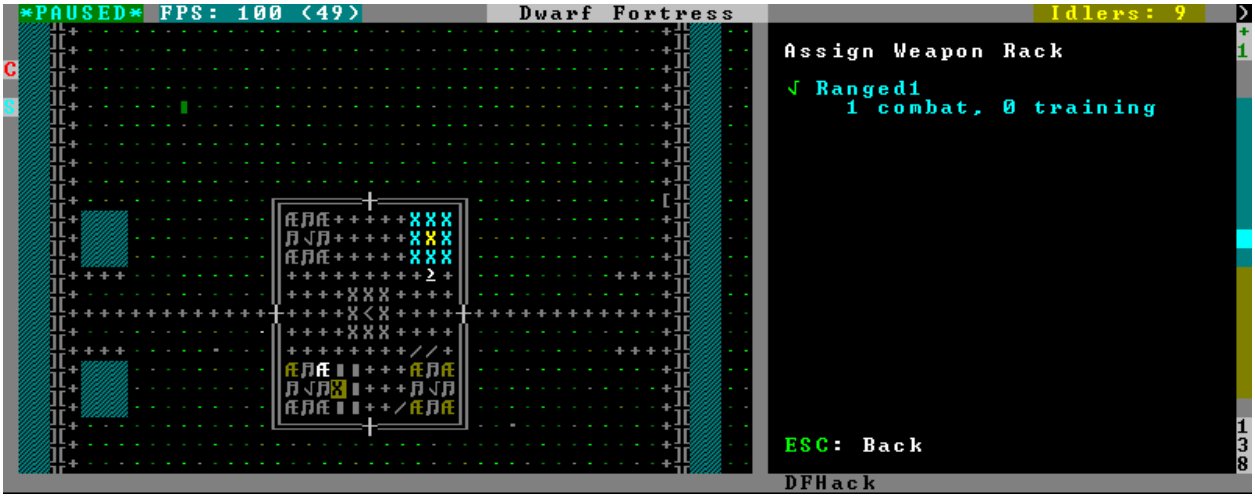
Contrary to the common misconception, all these uses are controlled by the *Individual Equipment* usage flag. The *Squad Equipment* flag is actually intended for ammo, but the game does even less in that area than for armor and weapons. This plugin implements the following rules almost from scratch:

- 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.