DFHack Documentation

Release 0.47.05-r8

The DFHack Team

2022-12-02

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*
- *Getting help*
- **Source code** (**important:** read *Compiling DFHack* before attempting to build from source.)

Part II

User Manual

CHAPTER

ONE

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 DFHack tools 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 licenses*, 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*, or 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.

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.

1.2 Getting help

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

TWO

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 folders, including a `hack` folder where DFHack binary and system data is stored, a `dfhack-config` folder where user data and configuration is stored, and a `blueprints` folder where *quickfort* blueprints are stored. 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

THREE

DFHACK CORE

Contents

3.1 Command implementation

DFHack commands can be implemented in any of three ways:

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.

All tools distributed with DFHack are documented here.

3.2 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

3.2.1 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"
```

3.2.2 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*.

3.3 Configuration files

Most 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.

3.3.1 Init files

DFHack allows users to automatically run commonly-used DFHack commands when DF is first loaded, when a world is loaded, when a map is loaded, when a map is unloaded, and when a world 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 `dfhack-config/init` subdirectory in the main DF directory
2. `data/save/`*`world`*`/raw`, where `world` is the current save, and
3. `data/save/`*`world`*`/raw/objects`

For each of those directories, all matching init files will be executed in alphabetical order.

Before running matched init scripts in any of those locations, the `dfhack-config/init/default.*` file that matches the event will be run to load DFHack defaults. Only the `dfhack-config/init` directory is checked for this file, not any `raw` directories. If you want DFHack to load without running any of its default configuration commands, edit the `dfhack-config/init/default.*` files and comment out the commands you see there.

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

dfhack*.init

On startup, DFHack looks for files of the form `dfhack*.init` (where `*` is a placeholder for any string, including the empty string).

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

onLoad*.init

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

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 bugfix-tag-index script to run on *repeat*.

onMapLoad*.init

When a map is loaded, either in adventure or fort mode, DFHack looks for files of the form `onMapLoad*.init`, where `*` can be any string, including the empty string.

These files are best used for commands that are only relevant once there is a game map loaded.

onMapUnload*.init and onUnload*.init

When a map or world is unloaded, DFHack looks for files of the form `onMapUnload*.init` or `onUnload*.init`, respectively.

Modders often use unload init scripts to disable tools which should not run after a modded save is unloaded.

raw/init.d/*.lua

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.

3.3.2 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*.

3.4 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.

3.5 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

FOUR

DFHACK TOOLS

DFHack has **a lot** of tools. This page attempts to make it clearer what they are, how they work, and how to find the ones you want.

Contents

4.1 What tools are and how they work

DFHack is a Dwarf Fortress memory access and modification framework, so DFHack tools normally access Dwarf Fortress internals and make some specific changes.

Some tools just make a targeted change when you run them, like *unforbid*, which scans through all your items and removes the `forbidden` flag from each of them.

Some tools need to be enabled, and then they run in the background and make changes to the game on your behalf, like *autobutcher*, which monitors your livestock population and automatically marks excess animals for butchering.

And some tools just exist to give you information that is otherwise hard to come by, like *gui/petitions*, which shows you the active petitions for guildhalls and temples that you have agreed to.

4.2 Finding the tool you need

DFHack tools are tagged with categories to make them easier to find. These categories are listed in the next few sections. Note that a tool can belong to more than one category. If you already know what you're looking for, try the search or Ctrl-F on this page. If you'd like to see the full list of tools in one flat list, please refer to the annotated index.

4.3 DFHack tools by game mode

- **adventure** Tools that are useful while in adventure mode. Note that some tools only tagged with "fort" might also work in adventure mode, but not always in expected ways. Feel free to experiment, though!
- **dfhack** Tools that you use to run DFHack commands or interact with the DFHack library. This tag also includes tools that help you manage the DF game itself (e.g. settings, saving, etc.)
- **embark** Tools that are useful while on the fort embark screen or while creating an adventurer.
- **fort** Tools that are useful while in fort mode.
- **legends** Tools that are useful while in legends mode.

4.4 DFHack tools by theme

- **armok** Tools that give you complete control over an aspect of the game or provide access to information that the game intentionally keeps hidden.
- **auto** Tools that run in the background and automatically manage routine, toilsome aspects of your fortress.
- **bugfix** Tools that fix specific bugs, either permanently or on-demand.
- **design** Tools that help you design your fort.
- **dev** Tools that are useful when developing scripts or mods.
- **fps** Tools that help you manage FPS drop.
- **gameplay** Tools that introduce new gameplay elements.
- **inspection** Tools that let you view information that is otherwise difficult to find.
- **productivity** Tools that help you do things that you could do manually, but using the tool is better and faster.

4.5 DFHack tools by what they affect

- **animals** Tools that interact with animals.
- **buildings** Tools that interact with buildings and furniture.
- **graphics** Tools that interact with game graphics.
- **interface** Tools that interact with or extend the DF user interface.
- **items** Tools that interact with in-game items.
- **jobs** Tools that interact with jobs.
- **labors** Tools that deal with labor assignment.
- **map** Tools that interact with the game map.
- **military** Tools that interact with the military.
- **plants** Tools that interact with trees, shrubs, and crops.
- **stockpiles** Tools that interact with stockpiles.
- **units** Tools that interact with units.
- **workorders** Tools that interact with workorders.

4.6 All DFHack tools alphabetically

4.6.1 3dveins

Tags: fort | gameplay | map

Command: `3dveins`

Rewrite layer veins to expand in 3D space.

Existing, flat veins are removed and new 3D veins that naturally span z-levels are generated in their place. The transformation preserves the mineral counts reported by *prospector*.

Usage

```
3dveins [verbose]
```

The `verbose` option prints out extra information to the console.

Example

```
3dveins
```

New veins are generated using natural-looking 3D Perlin noise in order to produce a layout that flows smoothly 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 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.

4.6.2 RemoteFortressReader

Tags: dev | graphics

Backend for Armok Vision.

Command: `RemoteFortressReader_version`

Print the loaded RemoteFortressReader version.

Command: `load-art-image-chunk`

Gets an art image chunk by index.

This plugin provides an API for realtime remote fortress visualization. See Armok Vision.

Usage

`RemoteFortressReader_version` Print the loaded RemoteFortressReader version.

`load-art-image-chunk <chunk id>` Gets an art image chunk by index, loading from disk if necessary.

4.6.3 adaptation

Tags: fort | armok | units

Command: `adaptation`

Adjust a unit's cave adaptation level.

View or set level of cavern adaptation for the selected unit or the whole fort.

Usage

```
adaptation show him|all
adaptation set him|all <value>
```

The `value` must be between 0 and 800,000 (inclusive), with higher numbers representing greater levels of cave adaptation.

Examples

`adaptation set all 0` Clear the cave adaptation levels for all dwarves.

4.6.4 add-recipe

Tags: adventure | fort | gameplay

Command: `add-recipe`

Add crafting recipes to a civ.

Civilizations pick randomly from a pool of possible recipes, which means not all civs get high boots, for instance. This script can help fix that. Only weapons, armor, and tools are currently supported; things such as instruments are not.

Usage

`add-recipe native` Add all crafting recipes that your civ could have chosen from its pool, but did not.

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

`add-recipe single <item token>` Add a single item by the given item token.

Example

`add-recipe single SHOES:ITEM_SHOES_BOOTS` Allow your civ to craft high boots.

4.6.5 add-spatter

Tags: adventure | fort | gameplay | items

Make tagged reactions produce contaminants.

Give some use to all those poisons that can be bought from caravans! The plugin automatically enables itself when you load a world with reactions that include names starting with `SPATTER_ADD_`, so there are no commands to run to use it. These reactions will then produce contaminants on items instead of improvements. The contaminants are immune to being washed away by water or destroyed by *cleaners*.

You must have a mod installed that adds the appropriate tokens in order for this plugin to do anything.

4.6.6 add-thought

Tags: fort | armok | units

Command: `add-thought`

Adds a thought to the selected unit.

Usage

`add-thought --gui [--unit <id>]` Allows you to choose the thought to apply to the selected (or specified) unit through a series of graphical prompts.

`add-thought [--unit <id>] [<options>]` Add a thought to the selected (or specified) unit.

Examples

`add-thought --gui` Add a thought to the unit currently selected in the UI.

`add-thought --unit 23142 --emotion GRATITUDE --thought GoodMeal --strength 1` Make unit 23142 feel a light sense of gratitude for eating a good meal.

Options

`--emotion <id>` Specifies an emotion for the unit to associate with the given thought. To see a list of possible emotions, run `:lua @df.emotion_type`. If not specified, defaults to `-1` (i.e. no emotion).

`--thought <id>` The thought. To see a list of possible thoughts, run `:lua @df.unit_thought_type`. If not specified, defaults to `180`, or `NeedsUnfulfilled`. The id could also be the name of a syndrome. To see a list of syndromes in your world, run `devel/query --table df.global.world.raws.syndromes.all --search syn_name --maxdepth 1`. The id is the numerical index and the syn_name field is the name.

`--subthought <id>` The subthought identifier. If the thought is the name of a syndrome, then the subthought should be the syndrome id. If not specified, defaults to `0` (which is what you want for most thought types).

`--strength <strength>` The strength of the emotion, corresponding to the strength of the need that this emotion might cause or fulfill. Common values for this are `1` (Slight need), `2` (Moderate need), `5` (Strong need), and `10` (Intense need). If not specified, defaults to `0`.

`--severity <severity>` If the thought is the name of a syndrome, then the severity will be used as the severity of the syndrome.

4.6.7 adv-fix-sleepers

Tags: adventure | bugfix | units

Command: `adv-fix-sleepers`

Fix units who refuse to awaken in adventure mode.

Use this tool if you encounter sleeping units who refuse to awaken regardless of talking to them, hitting them, or waiting so long you die of thirst (Bug 6798). If you come across one or more bugged sleepers in adventure mode, simply run the script and all nearby sleepers will be cured.

Usage

```
adv-fix-sleepers
```

4.6.8 adv-max-skills

Tags: adventure | embark | armok

Command: `adv-max-skills`

Raises adventurer stats to max.

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

Usage

```
adv-max-skills
```

4.6.9 adv-rumors

Tags: adventure | interface

Command: `adv-rumors`

Improves the rumors menu in adventure mode.

Keybinding: CtrlA in `dungeonmode/ConversationSpeak`

In adventure mode, start a conversation with someone and then run this tool to improve the "Bring up specific incident or rumor" menu. Specifically, this tool will:

- Move entries into a single line to improve readability
- Add a "slew" keyword for filtering, making it easy to find your kills and not your companions'
- Trim repetitive words from the text

Usage

```
adv-rumors
```

4.6.10 alias

Tags: dfhack

Command: `alias`

Configure helper aliases for 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.

Example

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

4.6.11 animal-control

Tags: fort | productivity | animals

Command: `animal-control`

Quickly view, butcher, or geld groups of animals.

Animal control is useful for browsing through your animals and deciding which to butcher or geld based on their stats.

Usage

```
animal-control [<selection options>] [<command options>]
```

Examples

`animal-control --all` View all your animals and whether they are marked for gelding or butchering.

`animal-control --race DOG --showstats` View extended info on your dogs.

`animal-control --markfor gelding --id 1988` Mark the specified unit for gelding.

`animal-control --gelded --markfor slaughter` Mark all gelded animals for slaughter.

`animal-control --gelded --markedfor slaughter --unmarkfor slaughter` Unmark all gelded animals for slaughter.

Selection options

These options are used to specify what animals you want to select. If an option calls for an `<action>`, valid actions are `slaughter` and `gelding`.

`--all` Selects all units. This is the default if no selection options are specified.

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

`--race <value>` Selects units which match the specified race. This can be the string name or the numeric race id. Run `animal-control --all` to see which races you have right now.

`--markedfor <action>` Selects units which have been marked for the given action.

`--notmarkedfor <action>` Selects units which have not been marked for the given action.

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

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

`--male` Selects male units.

`--female` Selects female units.

Command options

If no command option is specified, the default is to just list the matched animals with some basic information.

`--showstats` Displays physical attributes of the selected animals.

`--markfor <action>` Marks selected animals for the given action.

`--unmarkfor <action>` Unmarks selected animals for the given action.

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

4.6.12 armoks-blessing

Tags: fort | armok | units

Command: `armoks-blessing`

Bless units with superior stats and traits.

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.

Usage

`armoks-blessing` Adjust stats and personalities to an ideal for all dwarves. No skills will be modified.

`armoks-blessing all` In addition to the stat and personality adjustments, set all skills for all dwarves to legendary.

`armoks-blessing list` Prints list of all skills.

`armoks-blessing classes` Prints list of all skill classes (i.e. named groups of skills).

`armoks-blessing <skill name>` Set a specific skill for all dwarves to legendary.

`armoks-blessing <class name>` Set a specific class (group of skills) for all dwarves to legendary.

Examples

`armoks-blessing Medical` All dwarves will have all medical related skills set to legendary.

`armoks-blessing RANGED_COMBAT` All dwarves become legendary archers.

4.6.13 assign-attributes

Tags: fort | armok | units

Command: `assign-attributes`

Adjust physical and mental attributes.

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.

For more information on attributes, please see the wiki.

Usage

```
assign-attributes [--unit <id>] <options>
```

Please run:

```
devel/query --table df --maxdepth 1 --search [ physical_attribute_type mental_attribute_
↪type ]
```

to see the list of valid attribute tokens.

Example

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

This will reset all attributes to a neutral value and will then set the following values (the ranges will differ based on race; this example assumes a dwarf is selected in the UI):

- 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."

Options

`--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 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 `attributes` are specified, the unit attributes will be reset and then the listed attributes will be modified.

Tiers

The tier corresponds to the text that DF will use to describe a unit with attributes in that tier's range. For example, here is the mapping for the "Strength" attribute:

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

4.6.14 assign-beliefs

Tags: fort | armok | units

Command: `assign-beliefs`

Adjust a unit's beliefs and values.

Beliefs are defined with the belief token and a number from -3 to 3, which describes the different levels of belief strength, as explained on the wiki.

Usage

```
assign-beliefs [--unit <id>] <options>
```

Please run `devel/query --table df.value_type` to see the list of valid belief tokens.

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.

Options

`--unit <id>` The target unit ID. If not present, the currently selected unit will be the target.

`--beliefs [ <belief> <level> [<belief> <level> ...] ]` The list of the beliefs to modify and their levels. The valid belief tokens can be found in the Personality_trait (substitute any space with underscores). Levels range from -3 to 3. There must be a space before and after each square bracket.

`--reset` Reset all beliefs to a neutral level, aligned with the unit's race's cultural values. If both this option and `--beliefs` are specified, the unit's beliefs will be reset and then the listed beliefs will be modified.

Belief strengths

The belief strength corresponds to the text that DF will use to describe a unit's 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, which is dependent on the race of the unit.

4.6.15 assign-facets

Tags: fort | armok | units

Command: `assign-facets`

Adjust a unit's facets and traits.

Facets are defined with a token and a number from -3 to 3, which describes the different levels of facet strength, as explained on the wiki

Usage

```
assign-facets [--unit <id>] <options>
```

Please run `devel/query --table df.personality_facet_type` to see a list of valid facet tokens.

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.

Options

`--unit <id>` The target unit ID. If not present, the currently selected unit will be the target.

`--facets [ <facet> <level> [<facet> <level> ...] ]` The list of the facets to modify and their levels. The valid facet tokens can be found in the Personality_trait (substitute any space with underscores). Levels range from -3 to 3. There must be a space before and after each square bracket.

`--reset` Reset all facets to a neutral level, aligned with the unit's race's cultural values. If both this option and `--facets` are specified, the unit's facets will be reset and then the listed facets will be modified.

Facet strengths

The facet strength corresponds to the text that DF will use to describe a unit's 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, which is dependent on the race of the unit.

4.6.16 assign-goals

Tags: fort | armok | units

Command: `assign-goals`

Adjust a unit's goals and dreams.

Goals are defined with a goal token and a flag indicating whether the goal has been achieved. For now, this flag should always be set to false. For a list of possible goals, please run `devel/query --table df.goal_type` or see the wiki.

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

Usage

```
assign-goals [--unit <id>] <options>
```

Example

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

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

Options

`--unit <id>` The target unit ID. If not present, the currently selected unit will be the target.

`--goals [ <goal> false [<goal> false ...] ]` The list of goals to add. The valid goal tokens can be found in the Personality_trait (substitute any space with underscores). There must be a space before and after each square bracket.

`--reset` Clear all goals. If both this option and `--goals` are specified, the unit's goals will be cleared and then the listed goals will be added.

4.6.17 assign-minecarts

Tags: fort | productivity

Command: `assign-minecarts`

Assign minecarts to hauling routes.

This script allows you to assign minecarts to hauling routes without having to use the in-game interface.

Note that a hauling route must have at least one stop defined before a minecart can be assigned to it.

Usage

`assign-minecarts list` Print information about your hauling routes, including whether they currently have minecarts assigned to them.

`assign-minecarts all|<route id> [-q|--quiet]` Find and assign a free minecart to all hauling routes (or the specified hauling route). Hauling routes that already have a minecart assigned to them are skipped.

Add `-q` or `--quiet` to suppress extra informational output.

Example

```
assign-minecarts all
```

4.6.18 assign-preferences

Tags: fort | armok | units

Command: `assign-preferences`

Adjust a unit's preferences.

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, please see the wiki.

Note the last three types of preferences ("like poetic form", "like musical form", and "like dance form") are not supported by this script.

Usage

```
assign-goals [--unit <id>] <options>
```

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 strawberry 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
```

Options

For each of the parameters that take lists of tokens, if there is a space in the token name, please replace it with an underscore. Also, there must be a space before and after each square bracket. If only one value is provided, the square brackets can be omitted.

`--unit <id>` The target unit ID. If not present, the currently selected unit will be the target.

`--likematerial [ <token> [<token> ...] ]` This is usually set to three tokens: a type of stone, a type of metal, and a type of gem. It can also be a type of wood, glass, leather, horn, pearl, ivory, a decoration material - coral or amber, bone, shell, silk, yarn, or cloth. Please include the full tokens, not just a part.

`--likecreature [ <token> [<token> ...] ]` For this preference, you can just list the species as the token. For example, a creature token can be something like `CREATURE:SPARROW:SKIN`. Here, you can just say `SPARROW`.

`--likefood [ <token> [<token> ...] ]` This usually contains at least a type of alcohol. It can also be a type of meat, fish, cheese, edible plant, cookable plant/creature extract, cookable mill powder, cookable plant seed, or cookable plant leaf. Please write the full tokens.

`--hatecreature [ <token> [<token> ...] ]` As in `--likecreature` above, you can just list the species in the token. The creature should be a type of `HATEABLE` vermin which isn't already explicitly liked, but no check is performed to enforce this.

`--likeitem [ <token> [<token> ...] ]` This can be a kind of weapon, a kind of ammo, a 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/objects/" folder), then specify the items using the full tokens found in those files (see examples in this help).

`--likeplant [ <token> [<token> ...] ]` As in `--likecreature` above, you can just list the tree or plant species in the token.

`--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/objects/" folder). You can use the full token or just the color name.

`--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/objects/" folder). You can use the full token or just the shape name.

`--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.

4.6.19 assign-profile

Tags: fort | armok | units

Command: `assign-profile`

Adjust characteristics of a unit according to saved profiles.

This tool can load a profile stored in a JSON file and apply the characteristics to a unit.

A profile can describe which attributes, skills, preferences, beliefs, goals, and facets a unit will have. The script relies on the presence of the other `assign-...` modules in this collection; please refer to the other modules' documentation for more information on the kinds of characteristics you can set.

See the "hack/scripts/dwarf_profiles.json" file for examples of the json schema. Please make a new file with your own additions, though, since the example file will get overwritten when you upgrade DFHack.

Usage

```
assign-profile [--unit <id>] <options>
```

Examples

- Loads and applies the profile called "DOCTOR" in the default json file, resetting the characteristics that are changed by the profile:

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

- Loads and applies a profile called "ARCHER" in a file you (the player) wrote. It keeps all the old characteristics except 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 /dfhack-config/assign-profile/military_profiles.json --
  ↪profile ARCHER --reset [ ATTRIBUTES SKILLS ]
  ```

Options

`--unit <id>` The target unit ID. If not present, the currently selected unit will be the target.

`--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 inside the json file to apply.

`--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 characteristics, those characteristics will be reset, and then what is described in the profile will be applied. If set to the string `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 are: `ALL`, `PROFILE`, `ATTRIBUTES`, `SKILLS`, `PREFERENCES`, `BELIEFS`, `GOALS`, and `FACETS`. There must be a space before and after each square bracket. If only one value is provided, the square brackets can be omitted.

4.6.20 assign-skills

Tags: fort | armok | units

Command: `assign-skills`

Adjust a unit's skills.

Skills are defined by their token and their rank. You can see a list of valid skill tokens by running `devel/query --table df.job_skill` or by browsing the wiki.

Usage

```
assign-skills [--unit <id>] <options>
```

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).

Options

`--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 20 (legendary + 5). It is actually possible to go beyond 20 (no check is performed), but the effect on the game may not be predictable. 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 `--skills`, first all the unit skills will be cleared and then the listed skills will be added.

Skill ranks

Here is the mapping from rank value to description:

Rank	Rank description
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, please see: https://dwarffortresswiki.org/index.php/DF2014:Skill#Skill_level_names

4.6.21 autobutcher

Tags: fort | auto | fps | animals

Command: `autobutcher`

Automatically butcher excess livestock.

This plugin monitors how many pets you have of each gender and age and assigns excess livestock for slaughter. It requires that you add the target race(s) to a watch list. Units will be ignored if they are:

- Untamed
- 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 (defaults are: 1 male kid, 5 female kids, 1 male adult, 5 female 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.

Usage

`enable autobutcher` Start processing livestock according to the configuration. Note that no races are watched by default. You have to add the ones you want to monitor with `autobutcher watch`, `autobutcher target` or `autobutcher autowatch`.

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

`autobutcher noautowatch` Stop auto-adding new races to the watch list.

`autobutcher target <fk> <mk> <fa> <ma> all|new|<race> [<race> ...]` Set target counts for the specified races: - fk = number of female kids - mk = number of male kids - fa = number of female adults - ma = number of female adults If you specify `all`, then this command will set the counts for all races on your current watchlist (including the races which are currently set to 'unwatched') and sets the new default for future watch commands. If you specify `new`, then this command just sets the new default counts for future watch commands without changing your current watchlist. Otherwise, all space separated races listed will be modified (or added to the watchlist if they aren't there already).

`autobutcher ticks <ticks>` Change the number of ticks between scanning cycles when the plugin is enabled. By default, a cycle happens every 6000 ticks (about 8 game days).

`autobutcher watch all|<race> [<race> ...]` Start watching the listed races. If they aren't already in your watchlist, then they will be added with the default target counts. If you specify the keyword `all`, then all races in your watchlist that are currently marked as unwatched will become watched.

`autobutcher unwatch all|<race> [<race> ...]` Stop watching the specified race(s) (or all races on your watchlist if `all` is given). The current target settings will be remembered.

`autobutcher forget all|<race> [<race> ...]` Unwatch the specified race(s) (or all races on your watchlist if `all` is given) and forget target settings for it/them.

`autobutcher [list]` Print status and current settings, including the watchlist. This is the default command if autobutcher is run without parameters.

`autobutcher list_export` Print commands required to set the current settings in another fort.

To see a list of all races, run this command:

> devel/query –table df.global.world.raws.creatures.all –search ^creature_id –maxdepth 1

Though not all the races listed there are tameable/butcherable.

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

Keep at most 7 kids (4 female, 3 male) and at most 3 adults (2 female, 1 male) for turkeys. Once the kids grow up, the oldest adults will get slaughtered. Excess kids will get slaughtered starting the the youngest to allow that the older ones grow into adults:

```
autobutcher target 4 3 2 1 BIRD_TURKEY
```

Configure useful limits for dogs, cats, geese (for eggs, leather, and bones), alpacas, sheep, and llamas (for wool), and pigs (for milk and meat). All other unnamed tame units will be marked for slaughter as soon as they arrive in your fortress:

```
enable autobutcher
autobutcher target 2 2 2 2 DOG
autobutcher target 1 1 2 2 CAT
autobutcher target 50 50 14 2 BIRD_GOOSE
autobutcher target 2 2 4 2 ALPACA SHEEP LLAMA
autobutcher target 5 5 6 2 PIG
autobutcher target 0 0 0 0 new
autobutcher autowatch
```

4.6.22 autochop

Tags: fort | auto | plants

Auto-harvest trees when low on stockpiled logs.

This plugin can designate trees for chopping when your stocks are low on logs.

Usage

```
enable autochop
```

Then, open the settings menu with `c` from the designations menu (the option appears when you have "Chop Down Trees" selected with `d-t`).

Set your desired thresholds and enable autochopping with `a`.

You can also restrict autochopping to specific burrows. Highlight a burrow name with the Up/Down arrow keys and hit `Enter` to mark it as the autochop burrrow.

Autochop checks your stock of logs and designates trees once every in game day.

4.6.23 autoclothing

Tags: fort | auto | workorders

Command: `autoclothing`

Automatically manage clothing work orders.

This command allows you to set how many of each clothing type every citizen should have.

Usage

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

`material` can be "cloth", "silk", "yarn", or "leather". The `item` can be anything your civilization can produce, such as "dress" or "mitten".

When invoked without parameters, it shows a summary of all managed clothing orders. When invoked with a material and item, but without a quantity, it shows the current configuration for that material and item.

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.

4.6.24 autodump

Tags: fort | armok | fps | productivity | items | stockpiles

Automatically set items in a stockpile to be dumped.

Command: `autodump`

Teleports items marked for dumping to the cursor position.

Command: `autodump-destroy-here`

Destroy items marked for dumping under the cursor.

Keybinding: CtrlShiftK in `dwarfmode`

Command: `autodump-destroy-item`

Destroys the selected item.

Keybinding: CtrlK in `dwarfmode`

When *enabled*, this plugin adds an option to the `q` menu for stockpiles. When the `autodump` option is selected for the stockpile, any items placed in the stockpile will automatically be designated to be dumped.

When invoked as a command, it can instantly move all unforbidden items marked for dumping to the tile under the cursor. After moving the items, the dump flag is unset and the forbid flag is set, just as if it had been dumped normally. Be aware that dwarves that are en route to pick up the item for dumping may still come and move the item to your dump zone.

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

Usage

```
enable autodump
autodump [<options>]
autodump-destroy-here
autodump-destroy-item
```

`autodump-destroy-here` is an alias for `autodump destroy-here` and is intended for use as a keybinding.

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

Options

`destroy` Destroy instead of dumping. Doesn't require a cursor. If `autodump` is called again with this option before the game is resumed, it cancels pending destroy actions.

`destroy-here` Destroy items marked for dumping under the cursor.

`visible` Only process items that are not hidden.

`hidden` Only process hidden items.

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

Examples

`autodump` Teleports items marked for dumping to the cursor position.

`autodump destroy` Destroys all unforbidden items marked for dumping

`autodump-destroy-item` Destroys the selected item.

4.6.25 autofarm

Tags: fort | auto | plants

Command: `autofarm`

Automatically manage farm crop selection.

Periodically scan your plant stocks and assign crops to your farm plots based on which plant stocks are low (as long as you have the appropriate seeds). The target threshold for each crop type is configurable.

Usage

`enable autofarm` Enable the plugin and start managing crop assignment.

`autofarm runonce` Updates all farm plots once, without enabling the plugin.

`autofarm status` Prints status information, including any defined thresholds.

`autofarm default <number>` Sets the default threshold.

`autofarm threshold <number> <type> [<type> ...]` Sets thresholds of individual plant types.

You can find the identifiers for the crop types in your world by running the following command:

```
lua "for _,plant in ipairs(df.global.world.raws.plants.all) do if plant.flags.SEED then␣
↪print(plant.id) end end"
```

Examples

`autofarm default 30` Set the default threshold to 30.

`autofarm threshold 150 MUSHROOM_HELMET_PLUMP GRASS_TAIL_PIG` Set the threshold for Plump Helmets and Pig Tails to 150

4.6.26 autogems

Tags: fort | auto | workorders

Automatically cut rough gems.

Command: `autogems-reload`

Reloads the autogems configuration file.

Automatically cut rough gems. This plugin periodically scans your stocks of rough gems and creates manager orders for cutting them at a Jeweler's Workshop.

Usage

`enable autogems` Enables the plugin and starts autocutting gems according to its configuration.

`autogems-reload` Reloads the autogems configuration file. You might need to do this if you have manually modified the contents while the game is running.

Run *gui/autogems* for a configuration UI, or access the `Auto Cut Gems` option from the Current Workshop Orders screen (`o-W`).

4.6.27 autohauler

Tags: fort | auto | labors

Command: `autohauler`

Automatically manage hauling labors.

Similar to *autolabor*, but instead of managing all labors, autohauler only addresses hauling labors, leaving the assignment of skilled labors entirely up to you. You can use the in-game *manipulator* UI or an external tool like Dwarf Therapist to do so.

Idle dwarves who are not on active military duty 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 a few dwarves and almost every non-military dwarf having at least one skilled labor assigned.

Autohauler allows a skill to be used as a flag to exempt a dwarf from autohauler's effects. By default, this is the unused ALCHEMIST labor, but it can be changed by the user.

Autohauler uses DFHack's *debug* functionality to display information about the changes it makes. The amount of information displayed can be controlled through appropriate use of the `debugfilter` command.

Usage

`enable autohauler` Start managing hauling labors. This is normally all you need to do. Autohauler works well on default settings.

`autohauler status` Show autohauler status and status of fort dwarves.

`autohauler <labor> haulers` Set whether a particular labor should be assigned to haulers.

`autohauler <labor> allow|forbid` Set whether a particular labor should mark a dwarf as exempt from hauling. By default, only the `ALCHEMIST` labor is set to `forbid`.

`autohauler reset-all|<labor> reset` Reset a particular labor (or all labors) to their default haulers/allow/forbid state.

`autohauler list` Show the active configuration for all labors.

`autohauler frameskip <number>` Set the number of frames between runs of autohauler.

Examples

`autohauler HAUL_STONE haulers` Set stone hauling as a hauling labor (this is already the default).

`autohauler RECOVER_WOUNDED allow` Allow the "Recover wounded" labor to be manually assigned by the player. By default it is automatically given to haulers.

`autohauler MINE forbid` Don't assign hauling labors to dwarves with the Mining labor enabled.

4.6.28 autolabor

Tags: fort | auto | labors

Command: `autolabor`

Automatically manage dwarf labors.

Autolabor attempts to keep as many dwarves as possible busy while allowing dwarves to specialize in specific skills.

Autolabor frequently checks how many jobs of each type are available and sets labors proportionally in order to get them all done quickly. Labors with equipment – mining, hunting, and woodcutting – which are abandoned if labors change mid-job, are handled slightly differently to minimize churn.

Dwarves on active military duty or dwarves assigned to burrows are left untouched by autolabor.

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

Usage

```
enable autolabor
```

Anything beyond this is optional - autolabor works well with the default settings. Once you have enabled it in a fortress, it stays enabled until you explicitly disable it, even if you save and reload your game.

By default, each labor is assigned to between 1 and 200 dwarves (2-200 for mining). 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 bookkeeper 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, excluding those who can't do the job. Dwarves who are masters of a skill are deprioritized for other skills. The labor is then added to the best <minimum> dwarves for that labor, then to 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 dwarves when we reach the maximum allowed.

Autolabor uses DFHack's *debug* functionality to display information about the changes it makes. The amount of information displayed can be controlled through appropriate use of the `debugfilter` command.

Examples

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

`autolabor CUT_GEM 1 1` Keep exactly 1 dwarf with gem cutting 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.

Advanced usage

`autolabor list` List current status of all labors. Use this command to see the IDs for all labors.

`autolabor status` Show basic status information.

`autolabor <labor> <minimum> [<maximum>] [<talent pool>]` Set range of dwarves assigned to a labor, optionally specifying the size of the pool of most skilled dwarves that will ever be considered for this labor.

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

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

`autolabor reset-all|<labor> reset` Return a labor (or all labors) to the default handling.

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

4.6.29 autolabor-artisans

Tags: fort | auto | labors

Command: `autolabor-artisans`

Configures autolabor to produce artisan dwarves.

This script runs an *autolabor* command for all labors where skill level influences output quality (e.g. Carpentry, Stone detailing, Weaponsmithing, etc.). It automatically enables autolabor if it is not already enabled.

After running this tool, you can make further adjustments to autolabor configuration by running autolabor commands directly.

Usage

```
autolabor-artisans <minimum> <maximum> <talent pool>
```

Examples:

`autolabor-artisans 0 2 3` Only allows a maximum of 2 dwarves to have skill-dependent labors enabled at once, chosen from the talent pool of the top 3 dwarves for that skill.

4.6.30 automaterial

Tags: fort | design | productivity | buildings | map

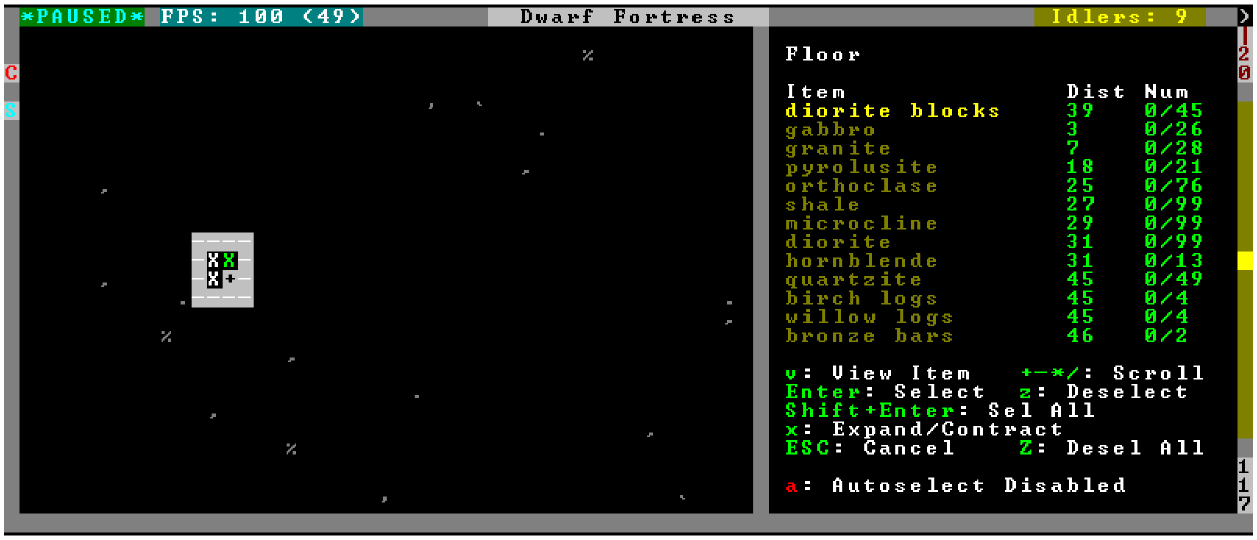
Sorts building materials by recent usage.

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

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 means you can place most constructions without having to search for your preferred material type.

Usage

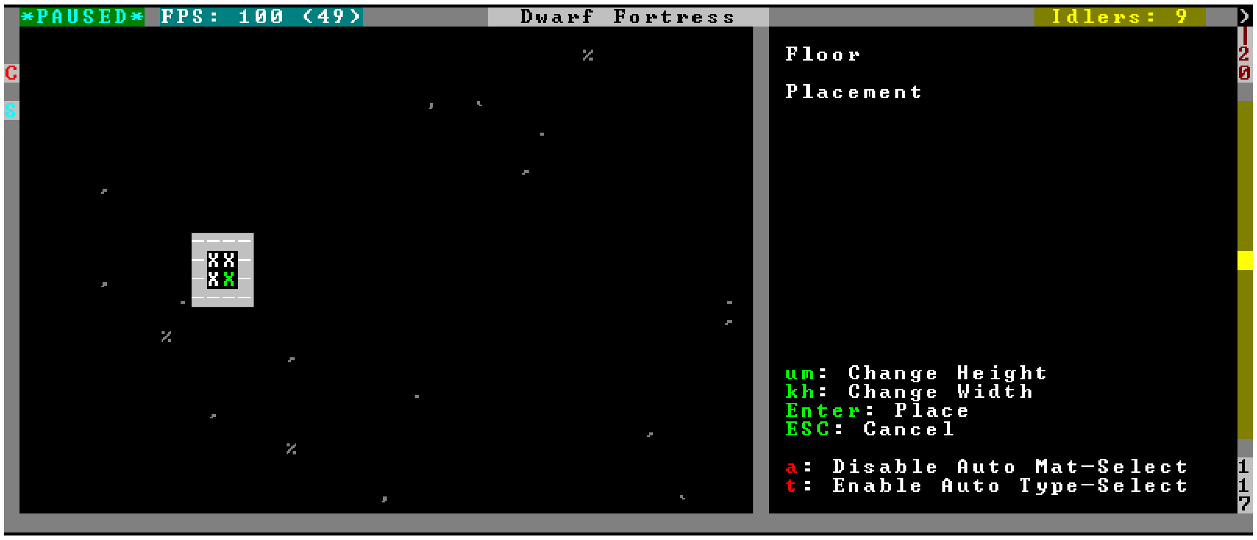
```
enable automaterial
```



```
*PAUSED* FPS: 100 (49)        Dwarf Fortress              Idlers: 9

Floor

Item                  Dist  Num
diorite blocks        39    0/45
gabbro                3     0/26
granite               7     0/28
pyrolusite            18    0/21
orthoclase            25    0/76
shale                 27    0/99
microcline            29    0/99
diorite               31    0/99
hornblende            31    0/13
quartzite             45    0/49
birch logs            45    0/4
willow logs           45    0/4
bronze bars           46    0/2

v: View Item      +-*/: Scroll
Enter: Select     z: Deselect
Shift+Enter: Sel All
x: Expand/Contract
ESC: Cancel       Z: Desel All

a: Autoselect Disabled
```

Pressing a while highlighting any material will enable that material for "auto select" for this construction type. You can enable multiple materials. 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 at all – the construction will be placed and you will be back in the construction menu.

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.

The `automaterial` plugin also enables extra construction placement modes, such as designating areas larger than 10x10 and allowing you to designate hollow rectangles instead of the default filled ones.

4.6.31 automelt

Tags: fort | productivity | items | stockpiles

Quickly designate items to be melted.

When *enabled*, this plugin adds an option to the `q` menu for stockpiles. When the `automelt` option is selected for the stockpile, any items placed in the stockpile will automatically be designated to be melted.

Usage

```
enable automelt
```

4.6.32 autonestbox

Tags: fort | auto | animals

Command: `autonestbox`

Auto-assign egg-laying female pets to nestbox zones.

To use this feature, you must create pen/pasture zones on the same tiles as built 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. Egg layers 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 egg layers could destroy your neat nestbox zones when they revert to wild.

Note that the age of the units is not checked, so you might get some egg-laying kids assigned to the nestbox zones. Most birds grow up quite fast, though, so they should be adults and laying eggs soon enough.

Usage

`enable autonestbox` Start checking for unpastured egg-layers and assigning them to nestbox zones.

`autonestbox` Print current status.

`autonestbox now` Run a scan and assignment cycle right now. Does not require that the plugin is enabled.

`autonestbox ticks <ticks>` Change the number of ticks between scan and assignment cycles when the plugin is enabled. The default is 6000 (about 8 days).

4.6.33 autonick

Tags: fort | productivity | units

Command: `autonick`

Give dwarves random unique nicknames.

Names are chosen randomly from the `dfhack-config/autonick.txt` config file, which you can edit with your own preferred names, if you like.

Dwarves who already have nicknames will keep the nicknames they have, and no other dwarf will be assigned that nickname.

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

Usage

```
autonick all [<options>]
```

You may wish to use this script with the “repeat” command so that new migrants automatically get nicknamed:

```
repeat -name autonick -time 3 -timeUnits months -command [ autonick all ]
```

Options

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

Config file format

The `dfhack-config/autonick.txt` config file has a simple format:

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

You can add any nicknames you like!

4.6.34 autotrade

Tags: fort | productivity | items | stockpiles

Quickly designate items to be traded.

When *enabled*, this plugin adds an option to the `q` menu for stockpiles. When the `autotrade` option is selected for the stockpile, any items placed in the stockpile will automatically be designated to be traded.

Usage

```
enable autotrade
```

4.6.35 autounsuspend

Tags: fort | auto | jobs

Command: `autounsuspend`

Keep construction jobs unsuspended.

This tool will unsuspend jobs that have become suspended due to inaccessible materials, items in the way, or worker dwarves getting scared by wildlife.

Also see *unsuspend* for one-time use.

Usage

```
autounsuspend start|stop
```

4.6.36 ban-cooking

Tags: fort | productivity | items | plants

Command: `ban-cooking`

Protect entire categories of ingredients from being cooked.

This tool provides a far more convenient way to ban cooking categories of foods than the native kitchen interface.

Usage

:: ban-cooking <type>

Valid types are `booze`, `brew`, `fruit`, `honey`, `milk`, `mill`, `oil`, `seeds` (i.e. non-tree plants with seeds), `tallow`, and `thread`.

Examples:

```
on-new-fortress ban-cooking booze; ban-cooking brew; ban-cooking fruit;
ban-cooking honey; ban-cooking milk; ban-cooking mill; ban-cooking oil;
ban-cooking seeds; ban-cooking tallow; ban-cooking thread
```

Ban cooking all otherwise useful ingredients once when starting a new fortress.

4.6.37 binpatch

Tags: dev

Command: `binpatch`

Applies or removes binary patches.

See *Patching the DF binary* for more info.

Usage

```
binpatch check|apply|remove <patchname>
```

4.6.38 blueprint

Tags: fort | design | buildings | map | stockpiles

Command: `blueprint`

Record a live game map in a quickfort blueprint.

With `blueprint`, you can export the structure of a portion of your fortress in a blueprint file that you (or anyone else) can later play back with *gui/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 `gui/blueprint` 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 GUI 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 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 coordinate 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.

`construct` Generate quickfort `#build` blueprints for constructions (e.g. flooring and walls).

`build` Generate quickfort `#build` blueprints for buildings (including furniture).

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

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

`query` Generate quickfort `#query` blueprints for configuring stockpiles and naming buildings.

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

`--nometa` *Meta blueprints* let you apply all blueprints that can be replayed at the same time (without unpausing the game) with a single command. This usually reduces the number of *quickfort* commands you need to run to rebuild your fort from about 6 to 2 or 3. If you would rather just have the low-level blueprints, this flag will prevent meta blueprints from being generated and any low-level blueprints from being *hidden* from the `quickfort list` command.

`-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"`.

`--smooth` Record all smooth tiles in the `smooth` phase. If this parameter is not specified, only tiles that will later be carved into fortifications or engraved will be smoothed.

`-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.

`group` Creates one file per group of blueprints that can be played back at the same time (without have to unpause the game and let dwarves fulfill jobs between blueprint runs).

`phase` Creates a separate file for each phase. Implies `--nometa` since meta blueprints can't combine blueprints that are in separate files.

4.6.39 bodyswap

Tags: adventure | armok | units

Command: `bodyswap`

Take direct control of any visible unit.

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.

Usage

```
bodyswap [--unit <id>]
```

If no specific unit id is specified, the target unit is the one selected in the user interface, such as by opening the unit's status screen or viewing its description.

Examples

`bodyswap` Takes control of the selected unit.

`bodyswap --unit 42` Takes control of unit with id 42.

4.6.40 brainwash

Tags: fort | armok | units

Command: `brainwash`

Set the personality of a dwarf to an ideal.

Modify the traits of the selected dwarf to match an idealized personality: as stable and reliable as possible to prevent tantrums even after months of misery.

Usage

:: brainwash <type>

Examples

`brainwash ideal` Sets the dwarf to have an ideal, stable personality

Types

The type is one of the following:

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

4.6.41 break-dance

Tags: fort | bugfix | units

Command: `break-dance`

Fixes buggy tavern dances.

Sometimes when a unit can't find a dance partner, the dance becomes stuck and never stops. This tool can get them unstuck.

Usage

```
break-dance
```

4.6.42 build-now

Tags: fort | armok | buildings

Command: `build-now`

Instantly completes building construction jobs.

By default, all unsuspended 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

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

4.6.43 building-hacks

Tags: fort | gameplay | buildings

Provides a Lua API for creating powered workshops.

See *building-hacks* for more details.

4.6.44 buildingplan

Tags: fort | design | buildings

Command: `buildingplan`

Plan building construction before you have materials.

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 manager work orders or *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. Materials are used to build the planned buildings as they are produced, with minimal space dedicated to stockpiles.

Usage

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

Running `buildingplan set` without parameters displays the current settings.

Global settings

The buildingplan plugin has global settings that can be set from the UI (`G` from any building placement screen, for example: `baG`). These settings can also be set via the `buildingplan set` command. The available settings are:

`all_enabled` (default: false) Enable planning mode for all building types.

`blocks`, `boulders`, `logs`, `bars` (defaults: true, true, true, false) Allow blocks, boulders, logs, or bars to be matched for generic "building material" items.

`quickfort_mode` (default: false) Enable compatibility mode for the legacy Python Quickfort (this setting is not required for DFHack *quickfort*)

The settings for `blocks`, `boulders`, `logs`, and `bars` are saved with your fort, so you only have to set them once and they will be persisted in your save.

If you normally embark with some blocks on hand for early workshops, you might want to add this line to your `dfhack-config/init/onMapLoad.init` file to always configure buildingplan to just use blocks for buildings and constructions:

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

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 this Quickfort mode to be enabled. The *quickfort* script will successfully integrate with buildingplan as long as the buildingplan plugin itself is enabled.

4.6.45 burial

Tags: fort | productivity | buildings

Command: `burial`

Configures all unowned coffins to allow burial.

Usage

```
burial [--pets]
```

if the `--pets` option is passed, coffins will also allow burial of pets.

Examples

`burial --pets` Configures all unowned coffins to allow burial, including pets.

4.6.46 burrows

Tags: fort | auto | design | productivity | map | units

Auto-expand burrows as you dig.

Command: `burrow`

Quickly add units/tiles to burrows.

When a wall inside a burrow with a name ending in + is dug out, the burrow will be extended to newly-revealed adjacent walls.

Usage

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

`burrow disable auto-grow` Disables auto-grow processing.

`burrow clear-unit <burrow> [<burrow> ...]` Remove all units from the named burrows.

`burrow clear-tiles <burrow> [<burrow> ...]` Remove all tiles from the named burrows.

`burrow set-units target-burrow <burrow> [<burrow> ...]` Clear all units from the target burrow, then add units from the named source burrows.

`burrow add-units target-burrow <burrow> [<burrow> ...]` Add units from the source burrows to the target.

`burrow remove-units target-burrow <burrow> [<burrow> ...]` Remove units in source burrows from the target.

`burrow set-tiles target-burrow <burrow> [<burrow> ...]` Clear target burrow tiles and add tiles from the names source burrows.

`burrow add-tiles target-burrow <burrow> [<burrow> ...]` Add tiles from the source burrows to the target.

`burrow remove-tiles target-burrow <burrow> [<burrow> ...]` Remove tiles in source burrows from the target.

In place of a source burrow, you can use one of the following keywords:

- `ABOVE_GROUND`
- `SUBTERRANEAN`
- `INSIDE`
- `OUTSIDE`
- `LIGHT`
- `DARK`
- `HIDDEN`
- `REVEALED`

to add tiles with the given properties.

4.6.47 cannibalism

Tags: adventure | gameplay

Command: `cannibalism`

Allows a player character to consume sapient corpses.

This tool clears the flag from items that mark them as being from a sapient creature. Use from an adventurer's inventory screen or an individual item's detail screen.

Usage

```
cannibalism
```

4.6.48 caravan

Tags: fort | armok | bugfix

Command: `caravan`

Adjust properties of caravans on the map.

This tool can help with caravans that are leaving too quickly, refuse to unload, or are just plain unhappy that you are such a poor negotiator.

Usage

```
caravan <command>
```

Also see *force* for creating caravans.

Examples

`caravan extend` Force a caravan that is leaving to return to the depot and extend their stay another 7 days.

`caravan unload` Fix a caravan that got spooked by wildlife and refuses to fully unload.

Commands

Commands listed with the argument `[<ids>]` can take multiple (space-separated) caravan IDs (see `caravan list`). If no IDs are specified, then the commands apply to all caravans on the map.

`list` List IDs and information about all caravans on the map.

`extend [<days> [<ids>]]` Extend the time that caravans stay at the depot by the specified number of days (defaults to 7). Also causes caravans to return to the depot if applicable.

`happy [<ids>]` Make 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.

`unload` Fix endless unloading at the depot. Run this if merchant pack animals were startled and now refuse to come to the trade depot.

4.6.49 catsplosion

Tags: fort | armok | animals

Command: `catsplosion`

Cause pregnancies.

This tool makes cats (or anything else) immediately pregnant. If you value your fps, it is a good idea to use this tool sparingly.

Usage

`catsplosion [<id> ...]` Makes animals with the given identifiers pregnant. Defaults to `CAT`.

`catsplosion list` List IDs of all animals on the map.

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

Examples

`catsplosion` Make all cats pregnant.

`catsplosion PIG SHEEP ALPACA` Get some quick butcherable meat.

`catsplosion DWARF` Have a population boom in your fort.

4.6.50 changeitem

Tags: adventure | fort | armok | items

Command: `changeitem`

Change item material or base quality.

By default, a change is only allowed if the existing and desired item materials are 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 forced changes can possibly result in items that crafters and haulers refuse to touch.

Usage

`changeitem info` Show details about the selected item. Does not change the item. You can use this command to discover RAW ids for existing items.

`changeitem [<options>]` Change the item selected in the `k` list or inside a container/inventory.

`changeitem here [<options>]` Change all items at the cursor position. Requires in-game cursor.

Examples

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

`changeitem q 5` Change currently selected item to masterpiece quality.

Options

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

`s`, `subtype <RAW id>` Change subtype. Must be followed by a valid subtype RAW id."

`q`, `quality <quality>` Change base quality. Must be followed by number (0-5) with 0 being no quality and 5 being masterpiece quality.

`force` Ignore subtypes and force the change to the new material.

4.6.51 changelayer

Tags: fort | armok | map

Command: `changelayer`

Change the material of an entire geology layer.

Note that one layer can stretch across many z-levels, and changes to the geology layer will affect all surrounding regions, not just your embark! Mineral veins and gem clusters will not be affected. Use *changevein* if you want to modify those.

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

Usage

```
changelayer <material RAW id> [<options>]
```

When run without options, `changelayer` will:

- only affect the geology layer at the current cursor position
- only affect the biome that covers the current cursor position
- not allow changing stone to soil and vice versa

You can use the *probe* command on various tiles around your map to find valid material RAW ids and to get an idea how layers and biomes are distributed.

Examples

`changelayer GRANITE` Convert the layer at the cursor position into granite.

`changelayer SILTY_CLAY force` Convert the layer at the 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 move the cursor to another tile. Then try again. If that doesn't help, then try to temporarily change some other layer, undo your changes, and try again for the layer you want to change. Saving and reloading your map also sometimes helps.
- 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 see some 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?

Options

`all_biomes` Change the corresponding geology layer for all biomes on your map. Be aware that the same geology layer can AND WILL be on different z-levels for different biomes.

`all_layers` Change all geology layers on your map (only for the selected biome unless `all_biomes` is also specified). 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 AND SAVE FIRST**. 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. Similarly, stone will not be magically replaced with soil, but you will get a soil floor after digging, so it could be helpful for creating farm plots on maps with no soil.

`verbose` Output details about what is being changed.

4.6.52 changevein

Tags: fort | armok | map

Command: `changevein`

Change the material of a mineral inclusion.

You can change a vein to any inorganic material RAW id. Note that this command only affects tiles within the current 16x16 block - for large veins and clusters, you will need to use this command multiple times.

You can use the *probe* command to discover the material RAW ids for existing veins that you want to duplicate.

Usage

```
changevein <material RAW id>
```

Example

`changevein NATIVE_PLATINUM` Convert vein at cursor position into platinum ore.

4.6.53 channel-safely

Tags: fort | auto

Command: `channel-safely`

Auto-manage channel designations to keep dwarves safe.

Multi-level channel projects can be dangerous, and managing the safety of your dwarves throughout the completion of such projects can be difficult and time consuming. This plugin keeps your dwarves safe (at least while channeling) so you don't have to. Now you can focus on designing your dwarven cities with the deep chasms they were meant to have.

Usage

```
enable channel-safely
channel-safely set <setting> <value>
channel-safely enable|disable <feature>
channel-safely <command>
```

When enabled the map will be scanned for channel designations which will be grouped together based on adjacency and z-level. These groups will then be analyzed for safety and designations deemed unsafe will be put into Marker Mode. Each time a channel designation is completed its group status is checked, and if the group is complete pending groups below are made active again.

Features and settings once set will persist until you change them, even if you save and reload your game.

Examples

`channel-safely` The plugin reports its configured status.

`channel-safely runonce` Runs the safety procedures once. You can use this if you prefer initiating scans manually.

`channel-safely disable require-vision` Allows the plugin to read all tiles, including the ones your dwarves know nothing about.

`channel-safely enable monitor` Enables monitoring active channel digging jobs. Meaning that if another unit it present or the tile below becomes open space the job will be paused or canceled (respectively).

`channel-safely set ignore-threshold 3` Configures the plugin to ignore designations equal to or above priority 3 designations.

Commands

runonce Run the safety procedures once to set the marker mode of designations.

rebuild Rebuild the designation group data. Intended for to be used in the event the marker mode isn't being set correctly (mostly for debugging).

Features

require-vision Toggle whether the dwarves need vision of a tile before channeling to it can be deemed unsafe. (default: enabled)

monitor Toggle whether to monitor the conditions of active digs. (default: disabled)

resurrect Toggle whether to resurrect units involved in cave-ins, and if monitor is enabled units who die while digging. (default: disabled)

insta-dig Toggle whether to use insta-digging on unreachable designations. Runs on the refresh cycles. (default: disabled)

Settings

refresh-freq The rate at which full refreshes are performed. This can be expensive if you're undertaking many mega projects. (default:600, twice a day)

monitor-freq The rate at which active jobs are monitored. (default:1)

ignore-threshold Sets the priority threshold below which designations are processed. You can set to 1 or 0 to effectively disable the scanning. (default: 5)

fall-threshold Sets the fall threshold beyond which is considered unsafe. (default: 1)

4.6.54 cleanconst

Tags: fort | fps | buildings

Command: `cleanconst`

Cleans up construction materials.

This tool 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 when you have many constructions on the map.

Usage

```
cleanconst
```

4.6.55 cleaners

Tags: adventure | fort | armok | fps | items | map | units

Provides commands for cleaning spatter from the map.

Command: `clean`

Removes contaminants.

Command: `spotclean`

Remove all contaminants from the tile under the cursor.

Keybinding: CtrlC

This plugin provides commands that clean the splatter that get scattered all over the map and that clings to your items and units. In an old fortress, cleaning with this tool can significantly reduce FPS lag! It can also spoil your !!FUN!!, so think before you use it.

Usage

```
clean all|map|items|units|plants [<options>]
spotclean
```

By default, cleaning the map leaves mud and snow alone. Note that cleaning units includes hostiles, and that cleaning items removes poisons from weapons.

`spotclean` works like `clean map snow mud`, removing all contaminants from the tile under the cursor. This is ideal if you just want to clean a specific tile but don't want the *cleaners* command to remove all the glorious blood from your entranceway.

Examples

`clean all` Clean everything that can be cleaned (except mud and snow).

`clean all mud item snow` Removes all spatter, including mud, leaves, and snow from map tiles.

Options

When cleaning the map, you can specify extra options for extra cleaning:

`mud` Also remove mud.

`item` Also remove item spatter, like fallen leaves and flowers.

`snow` Also remove snow coverings.

4.6.56 cleanowned

Tags: fort | productivity | items

Command: `cleanowned`

Confiscates and dumps garbage owned by dwarves.

This tool gets dwarves to give up ownership of scattered items and items with heavy wear and then marks those items for dumping. Now you can finally get your dwarves to give up their rotten food and tattered loincloths and go get new ones!

Usage

```
cleanowned [<types>] [dryrun]
```

When run without parameters, `cleanowned` will confiscate and dump rotten items and owned food that is left behind on the floor. Specify the `dryrun` parameter to just print out what would be done, but don't actually confiscate anything.

You can confiscate additional types of items by adding them to the commandline:

`scattered` Confiscate/dump all items scattered on the floor.

`x` Confiscate/dump items with wear level 'x' (lightly worn) and more.

`X` Confiscate/dump items with wear level 'X' (heavily worn) and more.

Or you can confiscate all owned items by specifying `all`.

Example

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

4.6.57 clear-smoke

Tags: fort | armok | fps | map

Command: `clear-smoke`

Removes all smoke from the map.

Note that this can leak memory and should be used sparingly.

Usage

```
clear-smoke
```

4.6.58 clear-webs

Tags: adventure | fort | armok | map | units

Command: `clear-webs`

Removes all webs from the map.

In addition to removing webs, this tool also frees any creatures who have been caught in one. Usable in both fortress and adventurer mode.

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

See also *fix/drop-webs*.

Usage

```
clear-webs [--unitsOnly|--websOnly]
```

Examples

`clear-webs` Remove all webs and free all webbed units.

Options

`--unitsOnly` Free all units from webs without actually removing any webs

`--websOnly` Remove all webs without freeing any units.

4.6.59 cls

Tags: dfhack

Command: `cls`

Clear the terminal screen.

Can also be invoked as `clear`. Note that this command does not delete command history. It just clears the text on the screen.

Usage

```
cls
```

4.6.60 colonies

Tags: fort | armok | map

Command: `colonies`

Manipulate vermin colonies and hives.

Usage

`colonies` List all vermin colonies on the map.

`colonies place [<type>]` Place a colony under the cursor.

`colonies convert [<type>]` Convert all existing colonies to the specified type.

The `place` and `convert` subcommands create or convert to honey bees by default.

Examples

`colonies place` Place a honey bee colony.

`colonies place ANT` Place an ant hive.

`colonies convert TERMITE` End your beekeeping industry by converting all colonies to termite mounds.

4.6.61 color-schemes

Tags: fort | gameplay | graphics

Command: `color-schemes`

Modify the colors used by the DF UI.

This tool allows you to set exactly which shades of colors should be used in the DF interface color palette.

To set up the colors, you must first create at least one file with color definitions inside. These files must be in the same format as `data/init/colors.txt` and contain RGB values for each of the color names. Just copy `colors.txt` and edit the values for your custom color schemes.

If you are interested in alternate color schemes, also see:

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

Usage

`color-schemes register <directory> [-f] [-q]` Register the directory (relative to the main DF game directory) where your color scheme files are stored.

`color-schemes list` List the color schemes from the registered directories.

`color-schemes default set <scheme name> [-q]` Set the named color scheme as the default. This value is stored so you only have to set it once, even if you start a new adventure/fort.

`color-schemes default load [-q]` Load the default color scheme that you previously set with `default set`.

`color-schemes load <scheme name> [-q]` Load the named color scheme.

Examples

Read your color scheme files from the `colorschemes` directory (a directory you created and populated with color scheme files) and set the default to the scheme named `mydefault`:

```
color-schemes register colorschemes
color-schemes default set mydefault
```

Read your color scheme files from the `colorschemes` directory (a directory you created and populated with color scheme files) and load the saved default. If you have a color scheme that you always want loaded, put these commands in your `dfhack-config/init/dfhack.init` file:

```
color-schemes -q register colorschemes
color-schemes default load
```

Options

-f, --force Register and read color schemes that are incomplete or are syntactically incorrect.

-q, --quiet Don't print any informational output.

API

When loaded as a module, this script will export the following functions:

- `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

4.6.62 combat-harden

Tags: fort | armok | military | units

Command: `combat-harden`

Set the combat-hardened value on a unit.

This tool can make a unit care more/less about seeing corpses.

Usage

```
combat-harden [<unit>] [<hardness>]
```

Examples

`combat-harden` Make the currently selected unit fully combat hardened

`combat-harden --citizens --tier 2` Make all fort citizens moderately combat hardened.

Unit options

`--all` All active units will be affected.

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

`--unit <id>` The given unit will be affected.

If no option is given or the indicated unit can't be found, the script will use the currently selected unit.

Hardness options

`--value <num>` A percent value (0 to 100, inclusive) to set combat hardened to.

`--tier <num>` 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 no option is given, the script defaults to using a value of 100.

4.6.63 combine-drinks

Tags: fort | productivity | items

Command: `combine-drinks`

Merge stacks of drinks in the selected stockpile.

Usage

```
combine-drinks [--max <num>] [--stockpile <id>]
```

Note that this command can only combine drinks that are in the same stockpile. It cannot combine drinks that are in different stockpiles or are not in a stockpile at all.

Examples

`combine-drinks` Combine drinks in the currently selected stockpile into as few barrels as possible, with a maximum of 30 units of drink per barrel.

`combine-drinks --max 100` Stuff more drinks into each barrel.

Options

`--max <num>` Set the maximum number of units of a drink type each barrel can contain. Defaults to 30.

`--stockpile <id>` The building id of the target stockpile. If not specified, the stockpile selected in the UI is used.

4.6.64 combine-plants

Tags: fort | productivity | items | plants

Command: `combine-plants`

Merge stacks of plants in the selected container or stockpile.

Usage

```
combine-plants [<options>]
```

Note that this command can only combine plants and plant growths that are in the same stockpile or container. It cannot combine items that are in different stockpiles/containers or are loose on the ground.

Examples

`combine-plants` Combine drinks in the currently selected stockpile or container into as few stacks as possible, with a maximum of 12 units per stack.

`combine-plants --max 100` Increase the maximum stack size for fewer stacks.

Options

`--max <num>` Set the maximum number of units of a drink type each barrel can contain. Defaults to 30.

`--stockpile <id>` The building id of the target stockpile. If not specified, the stockpile selected in the UI is used.

`--container <id>` The item id of the target container. If not specified, and `--stockpile` is also not specified, then the container item selected in the UI is used.

4.6.65 command-prompt

Tags: dfhack

Command: `command-prompt`

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

Keybinding: CtrlShiftP

Usage

```
command-prompt [entry]
```

If called with parameters, it starts with that text in the command edit area. This is most useful for developers, who can set a keybinding to open a language interpreter for lua or Ruby by starting with the *:lua* or *:rb* portions of the command already filled in.

Also see *gui/quickcmd* for a different take on running commands from the UI.

4.6.66 confirm

Tags: fort | interface

Command: `confirm`

Adds confirmation dialogs for destructive actions.

Now you can get the chance to avoid seizing goods from traders or deleting a hauling route in case you hit the key accidentally.

Usage

`enable confirm`, `confirm enable all` Enable all confirmation options. Replace with `disable` to disable all.

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

When run without parameters, `confirm` will report which confirmation dialogs are currently enabled.

4.6.67 create-items

Tags: fort | armok | items

Command: `create-items`

Spawn items under the cursor.

This script is handy to create basic resources you need to get your fortress started.

Usage

```
create-items <category> list
create-items <category> <material> [<quantity>]
```

If a quantity is not specified, it defaults to 20.

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 web CREATURE:SPIDER_CAVE_GIANT:SILK
create-items bar CREATURE:CAT:SOAP
create-items bar adamantine
```

Categories

The currently supported item categories are: `boulder`, `bar`, `plant`, `log`, and `web`.

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

4.6.68 createitem

Tags: adventure | fort | armok | items

Command: `createitem`

Create arbitrary items.

You can create new items of any type and made of any material. 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`, then 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.

Usage

`createitem <item> <material> [<count>]` Create <count> copies (default is 1) of the specified item made out of the specified material.

`createitem inspect` Obtain the item and material tokens of an existing item. Its output can be used directly as arguments to `createitem` to create new matching items (as long as the item type is supported).

`createitem floor|item|building` Subsequently created items will be placed on the floor beneath the selected unit's, inside the selected item, or as part of the selected building.

Note: `createitem building` is good for loading traps, but if you use it with workshops, you will have to deconstruct the workshop to access the item.

Examples

`createitem GLOVES:ITEM_GLOVES_GAUNTLETS INORGANIC:STEEL 2` Create 2 pairs of steel gauntlets (that is, 2 left gauntlets and 2 right gauntlets).

`createitem WOOD PLANT_MAT:TOWER_CAP:WOOD 100` Create 100 tower-cap logs.

`createitem PLANT_GROWTH BILBERRY:FRUIT` Create a single bilberry.

For more examples, the wiki.

4.6.69 cursecheck

Tags: fort | armok | inspection | units

Command: `cursecheck`

Check for cursed creatures.

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

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

By default, you will just see the count of cursed creatures 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, etc.) 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".

Usage

```
cursecheck [<options>]
```

Examples

- **`cursecheck`** Display a count of cursed creatures on the map (or under the cursor).
- **`cursecheck detail all`** Give detailed info about all cursed creatures including deceased ones.
- **`cursecheck nick`** Give a nickname to all living/active cursed creatures.

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.

If you see any living/active creatures with a cursetype of "unknown", then it is most likely a new type of curse introduced by a mod.

Options

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

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

`ids` Print the creature and race IDs.

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

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

4.6.70 cxxrandom

Tags: dev

Provides a Lua API for random distributions.

See *cxxrandom* for details.

4.6.71 deathcause

Tags: fort | inspection | units

Command: `deathcause`

Find out the cause of death for a creature.

Select a body part on the ground in the `k` menu or a unit in the `u` unit list, and `deathcause` will detail the cause of death of the creature.

Usage

```
deathcause
```

4.6.72 debug

Tags: dev

Provides commands for controlling debug log verbosity.

Command: `debugfilter`

Configure verbosity of DFHack debug output.

Debug output is grouped by plugin name, category name, and verbosity level.

The verbosity levels are:

- **`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.

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 category [<plugin regex>] [<category regex>]` List available debug plugin and category names. If filters aren't given then all plugins/categories are matched. This command is a good way to test regex parameters before you pass them to `set`.

`debugfilter filter [<id>]` List active and passive debug print level changes. The optional `id` parameter is the id listed as the first column in the filter list. If `id` is given, then the command shows extended information for the given filter only.

`debugfilter set [<level>] [<plugin regex>] [<category regex>]` Create a new debug filter to set category verbosity levels. This filter will not be saved when the DF process exists or the plugin is unloaded.

`debugfilter set persistent [<level>] [<plugin regex>] [<category regex>]` Store the filter in the configuration file to until `unset` is used to remove it.

`debugfilter unset <id> [<id> ...]` Delete a space separated list of filters.

`debugfilter disable <id> [<id> ...]` Disable a space separated list of filters but keep it in the filter list.

`debugfilter enable <id> [<id> ...]` Enable a space separated list of filters.

`debugfilter header [enable] | [disable] [<element> ...]` Control which header metadata is shown along with each log message. Run it without parameters to see the list of configurable elements. Include an `enable` or `disable` keyword to change whether specific elements are shown.

Example

`debugfilter set Warning core script` Hide script execution log messages (e.g. "Loading script: dfhack-config/dfhack.init"), which are normally output at Info verbosity in the "core" plugin with the "script" category.

4.6.73 deep-embark

Tags: embark | fort | gameplay

Command: `deep-embark`

Start a fort deep underground.

Moves the starting units and equipment to a specified underground region upon embarking so you can start your fort from there.

Run this script while setting up an embark, any time before the embark welcome message appears.

Usage

`deep-embark --depth <layer> [<options>]` Start monitoring the game for the welcome message. Once the embark welcome message appears, your units and equipment will automatically be moved to the specified layer.

`deep-embark --clear` Stop monitoring the game for the welcome message, effectively restoring normal embarks on the surface.

Example

`deep-embark --depth CAVERN_2` Embark in the second cavern layer

`deep-embark --depth UNDERWORLD --blockDemons` Embark in the underworld and disable the usual welcoming party.

Options

`--depth <layer>` Embark at the specified layer. Valid layers are: `CAVERN_1`, `CAVERN_2`, `CAVERN_3`, and `UNDERWORLD`.

`--blockDemons` Prevent the demon surge that is normally generated when you breach an underworld spire. Use this with `--depth UNDERWORLD` to survive past the first few minutes. Note that "wildlife" demon spawning will be unaffected.

`--atReclaim` Enable deep embarks when reclaiming sites.

`--clear` Re-enable normal surface embarks.

Deep embarks for mods

If you are creating a mod and you want to enable deep embarks by default, create a file called "onLoad.init" in the DF raw folder (if one does not exist already) and enter the `deep-embark` command within it.

4.6.74 deramp

Tags: fort | armok | map

Command: `deramp`

Removes all ramps designated for removal from the map.

It also removes any "floating" down ramps that can remain after a cave-in.

Usage

```
deramp
```

4.6.75 deteriorate

Tags: fort | auto | fps | gameplay | items | plants

Command: `deteriorate`

Cause corpses, clothes, and/or food to rot away over time.

When enabled, this script will cause the specified item types to slowly rot away. By default, items disappear after a few months, but you can choose to slow this down or even make things rot away instantly!

Now all those slightly worn wool shoes that dwarves scatter all over the place or the toes, teeth, fingers, and limbs from the last undead siege 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

`deteriorate start --types <types> [--freq <frequency>] [--quiet]` Starts deteriorating the specified item types while you play.

`deteriorate stop --types <types>` Stops deteriorating the specified item types.

`deteriorate status` Shows the item types that are currently being monitored and their deterioration frequencies.

`deteriorate now --types <types> [--quiet]` Causes all items (of the specified item types) to rot away within a few ticks.

You can have different types of items rotting away at different rates by running `deteriorate start` multiple times with different options.

Examples

Start deteriorating corpses and body parts:

```
deteriorate start --types corpses
```

Start deteriorating corpses and food and do it at twice the default rate:

```
deteriorate start --types corpses,food --freq 0.5,days
```

Deteriorate corpses quickly but clothes slowly:

```
deteriorate start -tcorpses -f0.1
deteriorate start -tclothes -f3,months
```

Options

`-f, --freq, --frequency <number>[,<timeunits>]` How often to increment the wear counters. `<timeunits>` can be one of `days`, `months`, or `years` and defaults to `days` if not specified. The default frequency of 1 day will result in items disappearing after several months. The number does not need to be a whole number. E.g. `--freq=0.5,days` is perfectly valid.

`-q, --quiet` Silence non-error output.

`-t, --types <types>` The comma-separated list of item types to affect. This option is required for `start`, `stop`, and `now` commands.

Types

clothes All clothing pieces that have an armor rating of 0 and are lying on the ground.

corpses All non-dwarf corpses and body parts. This includes potentially useful remains such as hair, wool, hooves, bones, and skulls. Use them before you lose them!

food All food and plants, regardless of whether they are in barrels or stockpiles. Seeds are left untouched.

4.6.76 die

Tags: dfhack

Command: `die`

Instantly exit DF without saving.

Use to exit DF quickly and safely.

Usage

```
die
```

4.6.77 dig

Tags: fort | design | productivity | map

Provides commands for designating tiles for digging.

Command: `digv`

Designate all of the selected vein for digging.

Keybinding: CtrlV in `dwarfmode`

Keybinding: CtrlShiftV -> `"digv x"` in `dwarfmode`

Command: `digvx`

Dig a vein across z-levels, digging stairs as needed.

Command: `digl`

Dig all of the selected layer stone.

Command: `diglx`

Dig layer stone across z-levels, digging stairs as needed.

Command: `digcircle`

Designate circles.

Command: `digtype`

Designate all vein tiles of the same type as the selected tile.

Command: `digexp`

Designate dig patterns for exploratory mining.

This plugin provides commands to make complicated dig patterns easy.

Usage

`digv [x] [-p<number>]` Designate all of the selected vein for digging.

`digvx [-p<number>]` Dig a vein across z-levels, digging stairs as needed. This is an alias for `digv x`.

`digl [x] [undo] [-p<number>]` Dig all of the selected layer stone. If `undo` is specified, removes the designation instead (for if you accidentally set 50 levels at once).

`diglx [-p<number>]` Dig layer stone across z-levels, digging stairs as needed. This is an alias for `digl x`.

`digcircle [<diameter>] [<solidity>] [<action>] [<designation>] [-p<number>]` Designate circles. The diameter is the number of tiles across the center of the circle that you want to dig. See the *digcircle* section below for options.

`digtype [<designation>] [-p<number>] [-z]` Designate all vein tiles of the same type as the selected tile. See the *digtype* section below for options.

`digexp [<pattern>] [<filter>] [-p<number>]` Designate dig patterns for exploratory mining. See the *digexp* section below for options.

All commands support specifying the priority of the dig designations with `-p<number>`, where the number is from 1 to 7. If a priority is not specified, the priority selected in-game is used as the default.

Examples

`digcircle filled 3 -p2` Dig a filled circle with a diameter of 3 tiles at dig priority 2.

`digcircle` Do it again (previous parameters are reused).

`expdig diag5 hidden` Designate the diagonal 5 pattern over all hidden tiles on the current z-level.

`expdig ladder designated` Take existing designations on the current z-level and replace them with the ladder pattern.

`expdig` Do it again (previous parameters are reused).

digcircle

The `digcircle` command can accept up to one option of each type below.

Solidity options:

`hollow` Designates hollow circles (default).

`filled` Designates filled circles.

Action options:

`set` Set designation (default).

`unset` Unset current designation.

`invert` Invert designations already present.

Designation options:

`dig` Normal digging designation (default).

`ramp` Dig ramps.

`ustair` Dig up staircases.

`dstair` Dig down staircases.

`xstair` Dig up/down staircases.

`chan` Dig channels.

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

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.

Designation options:

`dig` Normal digging designation.

`channel` Dig channels.

`ramp` Dig ramps.

`updown` Dig up/down staircases.

`up` Dig up staircases.

`down` Dig down staircases.

`clear` Clear any designations.

You can also pass a `-z` option, which restricts designations to the current z-level and down. This is useful when you don't want to designate tiles on the same z-levels as your carefully dug fort above.

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` The diag5 pattern rotated 90 degrees.

`ladder` A 'ladder' pattern.

`ladderr` The ladder pattern rotated 90 degrees.

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

`clear` Just remove all dig designations.

Filters:

`hidden` Designate only hidden tiles of z-level (default)

`all` Designate the whole z-level.

`designated` Take current designation and apply the selected pattern to it.

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

4.6.78 dig-now

Tags: fort | armok | map

Command: `dig-now`

Instantly complete dig designations.

This tool will magically complete 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. You can use the *position* command to get the current cursor position if you need it.

Examples

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

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

`dig-now --dump here` Dig tiles and teleport all generated boulders, ores, and gems to 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`.

`-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.

4.6.79 digFlood

Tags: fort | auto | map

Command: `digFlood`

Digs out veins as they are discovered.

Once you register specific vein types, this tool will automatically designate tiles of those types of veins for digging as your miners complete adjacent mining jobs. Note that it will *only* dig out tiles that are adjacent to a just-finished dig job, so if you want to autodig a vein that has already been discovered, you may need to manually designate one tile of the tile for digging to get started.

Usage

`enable digflood` Enable the plugin.

`digflood 1 <vein type> [<vein type> ...]` Start monitoring for the specified vein types.

`digFlood 0 <vein type> [<vein type> ...] 1` Stop monitoring for the specified vein types. Note the required 1 at the end.

`digFlood CLEAR` Remove all inorganics from monitoring.

`digFlood digAll1` Ignore the monitor list and dig any vein.

`digFlood digAll0` Disable digAll mode.

You can get the list of valid vein types with this command:

```
lua "for i,mat in ipairs(df.global.world.raws.inorganics) do if mat.material.flags.IS_
↪STONE and not mat.material.flags.NO_STONE_STOCKPILE then print(i, mat.id) end end"
```

Examples

`digFlood 1 MICROCLINE COAL_BITUMINOUS` Automatically dig microcline and bituminous coal veins.

`digFlood 0 MICROCLINE 1` Stop automatically digging microcline.

4.6.80 diggingInvaders

Tags: fort | gameplay | military | units

Command: `diggingInvaders`

Invaders dig and destroy to get to your dwarves.

Usage

`enable diggingInvaders` Enable the plugin.

`diggingInvaders add <race>` Register the specified race as a digging invader.

`diggingInvaders remove <race>` Unregisters the specified race as a digging invader.

`diggingInvaders now` Makes invaders try to dig now (if the plugin is enabled).

`diggingInvaders clear` Clears the registry of digging invader races.

`diggingInvaders edgesPerTick <n>` Makes the pathfinding algorithm work on at most n edges per tick. Set to 0 or lower to make it unlimited.

`diggingInvaders setCost <race> <action> <n>` Set the pathing cost per tile for a particular action. This determines what invaders consider to be the shortest path to their target.

`diggingInvaders setDelay <race> <action> <n>` Set the time cost (in ticks) for performing a particular action. This determines how long it takes for invaders to get to their target.

Note that the race is case-sensitive. You can get a list of races for your world with this command:

```
devel/query --table df.global.world.raws.creatures.all --search creature_id --maxdepth 1␣
↪--maxlength 5000
```

but in general, the race is what you'd expect, just capitalized (e.g. `GOBLIN` or `ELF`).

Actions:

`walk` Default cost: 1, default delay: 0. This is the base cost for the pathing algorithm.

`destroyBuilding` Default cost: 2, default delay: 1,000,

`dig` Default cost: 10,000, default delay: 1,000. This is for digging soil or natural stone.

`destroyRoughConstruction` Default cost: 1,000, default delay: 1,000. This is for destroying constructions made from boulders.

`destroySmoothConstruction` Default cost: 100, default delay: 100. This is for destroying constructions made from blocks or bars.

Example

`diggingInvaders add GOBLIN` Registers members of the GOBLIN race as a digging invader.

4.6.81 disable

Tags: dfhack

Command: `disable`

Deactivate a DFHack tool that has some persistent effect.

See the *enable* command for more info.

Usage

```
disable <plugin> [<plugin> ...]
```

4.6.82 do-job-now

Tags: fort | productivity | jobs

Command: `do-job-now`

Mark the job related to what you're looking at as high priority.

Keybinding: AltN in `dwarfmode|job|joblist|unit|unitlist|joblist|dungeon_monsterstatus|layer_unit_relations`

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 the job as high priority.

Apart from jobs that are queued from buildings, there is normally no visual indicator that the job is now high priority. If you use `do-job-now` from the keybinding, you have to check the dfhack console for output to see if the command succeeded.

If a work order is selected, every job currently active from this work order is adjusted, but not the future ones.

Also see the `do-job-now` *tweak*, which allows you to adjust job priorities from the `` j`obs screen, and `prioritize ``, which can automatically adjust priorities based on the type of job.

Usage

```
do-job-now
```

4.6.83 drain-aquifer

Tags: fort | armok | map

Command: `drain-aquifer`

Remove all aquifers on the map.

This tool irreversibly removes all 'aquifer' tags from the map blocks.

Usage

```
drain-aquifer
```

4.6.84 dwarf-op

Tags: fort | armok | units

Command: `dwarf-op`

Tune units to perform underrepresented job roles in your fortress.

`dwarf-op` examines the distribution of skills and attributes across the dwarves in your fortress and can rewrite the characteristics of a dwarf (or group of dwarves) so that they are fit to excel at the jobs that your current dwarves don't adequately cover.

It uses a library of profiles to define job classes, and generates dwarves with random variation so each dwarf is unique.

`dwarf-op` can also be used in a mode more similar to *assign-profile*, where you can specify precisely what archetype you want a for given dwarf, and `dwarf-op` can generate a random dwarf that matches that archetype.

Usage

```
dwarf-op --list <table>
dwarf-op --reset|--resetall
dwarf-op [--select <criteria>] <commands>
```

Examples

`dwarf-op --select unoptimized --clear --optimize` Transform newly arrived dwarves into the workers that your fort needs most.

`dwarf-op --select all --clear --optimize` Rebalance the distribution of skills and attributes across your units.

`dwarf-op --select names,Einstein --applytypes genius3,intuitive3,creative2,spaceaware3` Make dwarves with Einstein in their name into geniuses capable of surpassing the real Einstein.

`dwarf-op --select waves,2,3,5,7,11,13 --applyjobs MINER` Make all migrants in waves 2, 3, 5, 7, 11, and 13 very good miners.

`dwarf-op --select jobs,Stoneworker --applytypes fast3,strong5` Boost the speed and strength of your masons so they can carry boulders to their workshop faster.

Selection criteria

Note that dwarves whose name or profession starts with `.` or `,` are considered "protected", and will not be matched by the selection criteria options below unless specifically noted.

You can prepend the letter `p` to any option to include protected dwarves in your selection. For example, to truly select all dwarves, specify `pall` instead of `all`.

`highlighted` Selects only the in-game highlighted dwarf (from any screen), regardless of protection status. This is the default if no `--select` option is specified.

`all` Selects all dwarves.

`named` Selects dwarves with user-given names.

`unnamed` Selects dwarves without user-given names.

`employed`` Selects dwarves with custom professions. Does not include optimized dwarves.

`optimized` Selects dwarves that have been previously optimized by `dwarf-op`.

`unoptimized` Selects any dwarves that have not been previously optimized by `dwarf-op`.

`protected` Selects protected dwarves.

`unprotected` Selects unprotected dwarves.

`drunks` Selects any dwarves who have the `DRUNK` profession, including those who have been zeroed by the `--clear` command option.

`names,<name>[,<name>...]` Selects any dwarf with <name> anywhere in their name or nickname. This option ignores protection status.

`jobs,<job>[,<job>...]` Selects any dwarves with the specified custom professions.

`waves,<num>[,<num>...]` 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 them.

Options

`--reset` Forget which dwarves have been optimized. However, if you reload an existing save, the optimization list will be reloaded.

`--resetall` Forget which dwarves have been optimized and remove the saved optimization data.

`--list <table name>` Show the raw data tables that `dwarf-op` uses to make decisions. See the *Data tables* section below for which tables are available.

`--select <criteria>` Select a specific subset of dwarves that the specified *Command options* should act on.

Command options

`--show` Lists the selected dwarves. Useful for previewing selected dwarves before modifying them or looking up the migration wave number for a group.

`--clean <value>` Checks for skills with a rating of `<value>` and removes them from the dwarf's skill list.

`--clear` Zeroes the skills and attributes of selected dwarves. No attributes, no labors. Assigns `DRUNK` profession.

`--reroll [inclusive]` Clears attributes of selected dwarves, then rerolls that dwarf based on their jobs. Run `dwarf-op --list attrib_levels` to see how stats are distributed. If `inclusive` is specified, then attributes are not cleared, but rather will only be changed if the current reroll is better. This command ignores dwarves with jobs that are not listed in the `jobs` table.

`--optimize` Performs a job search for unoptimized dwarves. Run `dwarf-op --list job_distribution` to see how jobs are distributed.

`--applyjobs <job>[,<job>...]` Applies the listed jobs to the selected dwarves. Run `dwarf-op --list jobs` to see available jobs.

`--applyprofessions <profession>[,<profession>...]` Applies the listed professions to the selected dwarves. Run `dwarf-op --list professions` to see available professions.

`--applytypes <profession>[,<profession>...]` Applies the listed types to the selected dwarves. Run `dwarf-op --list dwf_types` to see available types.

`--renamejob <name>` Renames the selected dwarves' custom professions to the specified name.

Data tables

The data tables that `dwarf-op` uses are described below. They can be inspected with `dwarf-op --list <table name>`.

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

`attrib_levels` Defines stat distributions for both physical and mental attributes. Each level has a probability (p-value, or p) which indicates how likely a level will be used for a particular stat, e.g. strength or spacial awareness. The levels range from incompetent to unbelievable (god-like) and are mostly in line with what the game uses already. `dwarf-op` adds one additional level to push the unbelievable even higher, though.

In addition to a bell shaped p-value curve for the levels, there is additionally a standard deviation used to generate the value once a level has been selected, this makes the bell curve not so bell shaped in the end. Labors do not follow the same stat system and are more uniformly random, which are compensated for in the description of jobs/professions.

jobs Defines `dwarf-op`'s nameable jobs. Each job is comprised of required professions, optional professions, probabilities for each optional profession, a 'max' number of optional professions, and a list of types (from the `types` table below) to apply to dwarves in the defined job.

professions These are a subset of the professions DF has. All professions listed should 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 which match the skills built into Dwarf Fortress. Each skill is given a value which represents the bonus a dwarf will get for 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 and mental attributes, but can include skills as well. If it has skills listed, each skill will have a minimum and maximum value. The chosen values will be evenly distributed between these two numbers (inclusive).

Job specifications from the `jobs` table add these types to a dwarf to modify their stats. For the sake of randomness and individuality, each type has a probability for being additionally applied to a dwarf just by pure luck. This will bump some status up even higher than the base job calls for.

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

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

4.6.85 dwarfmonitor

Tags: fort | inspection | jobs | units

Command: `dwarfmonitor`

Report on dwarf preferences and efficiency.

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

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

It can also show heads-up display widgets with live fort statistics.

Usage

`enable dwarfmonitor` Enable tracking of job efficiency for display on the `dwarfmonitor stats` screen.

`dwarfmonitor stats` Show statistics and efficiency summary.

`dwarfmonitor prefs` Show a summary of preferences for dwarves in your fort.

Widget configuration

The following widgets are registered for display on the main fortress mode screen with the *overlay* framework:

`dwarfmonitor.cursor` Show the current keyboard and mouse cursor positions.

`dwarfmonitor.date` Show the in-game date.

`dwarfmonitor.misery` Show overall happiness levels of all dwarves.

`dwarfmonitor.weather` Show current weather (e.g. rain/snow).

They can be enabled or disable via the *overlay* command.

The `dfhack-config/dwarfmonitor.json` file can be edited to change widget configuration with any of the following fields:

- `date_format` (string): format for the `dwarfmonitor.date` widget:
    - `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.
- `coords_type` (string): the coordinate type to show in the `dwarfmonitor.cursor` widget:
    - `all` (the default): show all of the coordinate types listed here
    - `mouse_ui`: the X/Y UI coordinates of the tile the mouse is positioned over
    - `mouse_map`: the X/Y/Z map coordinates of the tile the mouse is positioned over (only if over the map)
    - `keyboard_map`: the X/Y/Z map coordinates of the tile selected by the keyboard-controlled `X` cursor in DF (if active)
- `coords_short` (boolean, default: `false`): if `true`, hides explanatory text from the `dwarfmonitor.cursor` widget, and only shows coordinates as `(X,Y,Z)`

Example configuration file:

```
{
    "date_format": "m/d/y",
    "coords_type": "mouse_map",
    "coords_short": false
}
```

4.6.86 dwarfvet

Tags: fort | gameplay | animals

Command: `dwarfvet`

Allows animals to be treated at animal hospitals.

Annoyed that your dragons become useless after a minor injury? Well, with dwarfvet, injured animals will be treated at an animal hospital, which is simply a hospital that is also an animal training zone. Dwarfs with the Animal Caretaker labor enabled will come to the hospital to treat animals. Normal medical skills are used (and trained), but no experience is given to the Animal Caretaker skill itself.

Usage

`enable dwarfvet` Enables the plugin.

`dwarfvet report` Reports all zones that the game considers animal hospitals.

`dwarfvet report-usage` Reports on animals currently being treated.

4.6.87 elevate-mental

Tags: fort | armok | units

Command: `elevate-mental`

Set mental attributes of a dwarf to an ideal.

Set all mental attributes of the selected dwarf to the maximum possible, or any value between 0 and 5000.

Usage

```
elevate-mental [value]
```

Examples

`elevate-mental` Boost mental attributes of the selected dwarf to the maximum.

`elevate-mental 100` Make the selected dwarf very stupid indeed.

4.6.88 elevate-physical

Tags: fort | armok | units

Command: `elevate-physical`

Set physical attributes of a dwarf to an ideal.

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

Usage

```
elevate-physical [value]
```

Examples

`elevate-physical` Boost physical attributes of the selected dwarf to the maximum.

`elevate-physical 100` Ensure your hammerer won't kill any citizens when they get punished.

4.6.89 embark-assistant

Tags: embark | fort | interface

Command: `embark-assistant`

Embark site selection support.

Run this command while the pre-embark screen is displayed to show extended (and reasonably correct) resource information for the embark rectangle as well as normally undisplayed sites in the current embark region. You will also have access to a site selection tool with far more options than DF's vanilla search tool.

If you enable the plugin, you'll also be able to invoke `embark-assistant` with the `A` key on the pre-embark screen.

Usage

```
enable embark-assistant
embark-assistant
```

Note the site selection tool requires a display height of at least 46 lines to display properly.

4.6.90 embark-skills

Tags: embark | fort | armok | units

Command: `embark-skills`

Adjust dwarves' skills when embarking.

When selecting starting skills for your dwarves on the embark screen, this tool can manipulate the skill values or adjust the number of points you have available to distribute.

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

Usage

`embark-skills points <N> [all]` Sets the skill points remaining of the selected dwarf (or all dwarves) to `N`.

`embark-skills max [all]` Sets all skills of the selected dwarf (or all dwarves) to "Proficient".

`embark-skills legendary [all]` Sets all skills of the selected dwarf (or all dwarves) to "Legendary".

Examples

`embark-skills points 10` After using all points for the selected dwarf, this will give you an extra 10 to assign to that dwarf.

`embark-skills legendary all` Make all your starting dwarves incredibly skilled.

4.6.91 embark-tools

Tags: embark | fort | interface

Command: `embark-tools`

Extend the embark screen functionality.

Usage

```
enable embark-tools
embark-tools enable|disable all
embark-tools enable|disable <tool> [<tool> ...]
```

Available tools are:

`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.

Example

`embark-tools enable all` Enable all embark screen extensions.

4.6.92 emigration

Tags: fort | auto | gameplay | units

Command: `emigration`

Allow dwarves to emigrate from the fortress when stressed.

If a dwarf is spiraling downward and is unable to cope in your fort, this tool will give them the choice to leave the fortress (and the map).

Dwarves will choose to leave in proportion to how badly stressed they are. Dwarves who can leave in friendly company (e.g. a dwarven merchant caravan) will choose to do so, but extremely stressed dwarves can choose to leave alone, or even in the company of a visiting elven bard!

The check is made monthly. A happy dwarf (i.e. with negative stress) will never emigrate.

Usage

```
emigration enable|disable
```

Example

`emigration enable` Start the emigration game mechanic.

4.6.93 empty-bin

Tags: fort | productivity | items

Command: `empty-bin`

Empty the contents of containers onto the floor.

This tool can quickly empty the contents of the selected container (bin, barrel, or pot) onto the floor, allowing you to access individual items that might otherwise be hard to get to.

Usage

```
empty-bin
```

4.6.94 enable

Tags: dfhack

Command: `enable`

Activate a DFHack tool that has some persistent effect.

Many plugins and scripts 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.

Usage

```
enable
enable <plugin> [<plugin> ...]
```

Examples

`enable manipulator` Enable the `manipulator` plugin.

`enable manipulator search` Enable multiple plugins at once.

4.6.95 eventful

Tags: dev | gameplay

Provides a Lua API for reacting to in-game events.

See *eventful* for details.

4.6.96 exportlegends

Tags: legends | inspection

Command: `exportlegends`

Exports legends data for external viewing.

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

When run from legends mode, you can export detailed data about your world so that it can be browsed with external programs like World Viewer and other similar utilities. The data exported with this tool is more detailed than what you can get with vanilla export functionality, and some external tools depend on this extra information.

`exportlegends` can be especially useful when you are generating a lot of worlds that you later want to inspect or when you want a map of every site when there are several hundred.

Usage

```
exportlegends <command> [<folder name>]
```

Valid commands are:

info Exports the world/gen info, the legends XML, and an extended info file.

custom Exports just the extended info file.

sites Exports all available site maps.

maps Exports all seventeen detailed maps.

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

The default folder name is generated from the region number of the world and the current in-world date: `legends-regionX-YYYYY-MM-DD`. You can use a different folder by naming it on the `exportlegends` command line. Nested paths are accepted, but all but the last folder has to already exist. To export to the top-level DF folder, specify `.` as the folder name.

Examples

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

`exportlegends all legends/myregion` Export all information to the `legends/myregion` folder.

`exportlegends custom .` Export just the extended info file to the DF folder (no subfolder).

4.6.97 exterminate

Tags: fort | armok | units

Command: `exterminate`

Kills things.

Kills any unit, or all units of a given race. You can target any unit on a revealed tile of the map, including ambushers, but caged/chained creatures are ignored.

Usage

`exterminate` List the available targets.

`exterminate this|him|her|it [magma|butcher]` Kills the selected unit.

`exterminate <race>[:<caste>] [magma|butcher]` Kills all available units of the specified race, or all undead units.

`exterminate undead [magma|butcher]` Kills all available undead units, regardless of race.

If `magma` is specified, a column of 7/7 magma is generated on top of the targets until they die. Warning: do not try this on magma-safe creatures! Also, using this mode on flyers is not recommended unless you like magma rain.

Alternately, if `butcher` is specified, `exterminate` will mark the units for butchering but does not kill them. A dwarf will take the creature to a butcher's shop and do the deed there. This mode is, of course, useful for pets and not for armed enemies.

Examples

`exterminate this` Kill the selected unit.

`exterminate` List the targets on your map.

`exterminate BIRD_RAVEN:male` Kill the ravens flying around the map (but only the male ones).

`exterminate undead magma` Kill all undead on the map by pouring magma on them.

Technical details

This tool kills by setting a unit's `blood_count` set to 0, which means immediate death at the next game tick. For creatures where this is not enough, such as vampires, it also sets animal.vanish_countdown to 2.

4.6.98 extinguish

Tags: fort | armok | buildings | items | map | units

Command: `extinguish`

Put out fires.

With this tool, you can put out fires affecting map tiles, plants, units, items, and buildings.

To select a target, place the cursor over it before running the script.

If your FPS is unplayably low because of the generated smoke, see *clear-smoke*.

Usage

`extinguish` Put out the fire under the cursor.

`extinguish --all` Put out all fires on the map.

`extinguish --location [ <x> <y> <z> ]` Put out the fire at the specified map coordinates. You can use the *position* tool to find out what the coordinates under the cursor are.

Examples

`extinguish --location [ 33 41 128 ]` Put out the fire burning on the surface at position x=33, y=41, z=128.

4.6.99 fastdwarf

Tags: fort | armok | units

Command: `fastdwarf`

Dwarves teleport and/or finish jobs instantly.

Usage

```
enable fastdwarf
fastdwarf <speed mode> [<tele mode>]
```

Examples

`fastdwarf 1` Make all your dwarves move and work at maximum speed.

`fastdwarf 1 1` In addition to working at maximum speed, dwarves also teleport to their destinations.

Options

Speed modes:

0 Dwarves move and work at normal rates.

1 Dwarves move and work at maximum speed.

2 ALL units move (and work) at maximum speed, including creatures and hostiles.

Tele modes:

0 No teleportation.

1 Dwarves teleport to their destinations.

4.6.100 feature

Tags: fort | armok | map

Command: `feature`

Control discovery flags for map features.

This tool allows you to toggle the flags that the game uses to track your discoveries of map features. For example, you can make the game think that you have discovered magma so that you can build magma workshops and furnaces. You can also toggle the cavern layer discovery flags so you can control whether trees, shrubs, and grass from the various cavern layers grow within your fortress.

Usage

`feature list` List all map features in your current embark by index.

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

`feature show <index>` Marks the indicated map feature as discovered.

`feature hide <index>` Marks the selected map feature as undiscovered.

There will usually be multiple features with the `subterranean_from_layer` type. These are the cavern layers, and they are listed in order from closest to the surface to closest to the underworld.

4.6.101 fillneeds

Tags: fort | armok | units

Command: `fillneeds`

Temporarily satisfy the needs of a unit.

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

Usage

`fillneeds` Make the selected unit focused and unstressed.

`fillneeds --unit <id>` Make the specified unit focused and unstressed.

`fillneeds --all` Make all units focused and unstressed.

4.6.102 filltraffic

Tags: fort | design | productivity | map

Command: `filltraffic`

Set traffic designations using flood-fill starting at the cursor.

Command: `alltraffic`

Set traffic designations for every single tile of the map.

Command: `restrictice`

Restrict traffic on all tiles on top of visible ice.

Command: `restrictliquids`

Restrict traffic on all visible tiles with liquid.

Usage

```
filltraffic <designation> [<options>]
alltraffic <designation>
restrictice
restrictliquids
```

For `filltraffic`, flood filling stops at walls and doors.

Examples

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

Options

Traffic designations:

H High Traffic.

N Normal Traffic.

L Low Traffic.

R Restricted Traffic.

Filltraffic extra options:

X Fill across z-levels.

B Include buildings and stockpiles.

P Include empty space.

4.6.103 firestarter

Tags: fort | armok | items | map | units

Command: `firestarter`

Lights things on fire.

Feel the need to burn something? Set items, locations, or even entire inventories on fire! Use while viewing an item, with the cursor over a map tile, or while viewing a unit's inventory.

Usage

```
firestarter
```

4.6.104 fix-ster

Tags: fort | armok | animals

Command: `fix-ster`

Toggle infertility for units.

Now you can restore fertility to infertile creatures or inflict infertility on creatures that you do not want to breed.

Usage

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

Specify `fert` or `ster` to indicate whether you want to make the target fertile or sterile, respectively.

If no additional options are given, the command affects only the currently selected unit.

Options

all Apply to all units on the map.

animals Apply to all non-dwarf creatures.

only:<race> Apply to creatures of the specified race.

Examples

fix-ster fert Make the selected unit fertile.

fix-ster fert all Ensure all units across the entire fort are fertile.

fix-ster ster only:DWARF Halt dwarven population growth.

4.6.105 fix-unit-occupancy

Tags: fort | bugfix | map

Command: `fix-unit-occupancy`

Fix phantom unit occupancy issues.

If you see “unit blocking tile” messages that you can’t account for (Bug 3499), this tool can help.

Usage

```
enable fix-unit-occupancy
fix-unit-occupancy [here] [-n]
fix-unit-occupancy interval <num_ticks>
```

When run without arguments (or with just the `here` or `-n` parameters), the fix just runs once. You can also have it run periodically by enabling the plugin.

Examples

fix-unit-occupancy Run once and fix all occupancy issues on the map.

fix-unit-occupancy -n Report on, but do not fix, all occupancy issues on the map.

Options

`here` Only operate on the tile at the cursor.

`-n` Report issues, but do not write any changes to the map.

`interval <num_ticks>` Set how often the plugin will check for and fix issues when it is enabled. The default is 1200 ticks, or 1 game day.

4.6.106 fixnaked

Tags: fort | armok | units

Command: `fixnaked`

Removes all unhappy thoughts due to lack of clothing.

If you're having trouble keeping your dwarves properly clothed and the stress is mounting, this tool can help you calm things down. `fixnaked` will go through each of your units, scan for unhappy thoughts due to lack of clothing, and remove the unhappy thoughts from your dwarves' minds.

Usage

```
fixnaked
```

4.6.107 fixveins

Tags: fort | bugfix | map

Command: `fixveins`

Restore missing mineral inclusions.

This tool can also remove invalid references to mineral inclusions if you broke your embark with tools like *tiletypes*.

Usage

```
fixveins
```

4.6.108 flashstep

Tags: adventure | armok

Command: `flashstep`

Teleport your adventurer to the cursor.

`flashstep` is a hotkey-friendly teleport that places your adventurer where your cursor is.

Usage

```
flashstep
```

4.6.109 flows

Tags: fort | inspection | map

Command: `flows`

Counts map blocks with flowing liquids.

If you suspect that your magma sea leaks into HFS, you can use this tool to be sure without revealing the map.

Usage

```
flows
```

4.6.110 follow

Tags: fort | interface | units

Command: `follow`

Make the screen follow the selected unit.

Once you exit from the current menu or cursor mode, the screen will stay centered on the unit. Handy for watching dwarves running around. Deactivated by moving the cursor manually.

Usage

```
follow
```

4.6.111 force

Tags: fort | armok | gameplay

Command: `force`

Trigger in-game events.

This tool triggers events like megabeasts, caravans, and migrants. Note that you can only trigger one caravan per civ at the same time, and that DF may choose to ignore events that are triggered too frequently.

Usage

```
force <event> [<civ id>]
```

The civ id is only used for `Diplomat` and `Caravan` events, and defaults to the player civilization if not specified.

The default civ IDs that you are likely to be interested in are:

- `MOUNTAIN` (dwarves)
- `PLAINS` (humans)
- `FOREST` (elves)

But to see IDs for all civilizations in your current game, run this command:

```
devel/query --table df.global.world.entities.all --search code --maxdepth 2
```

Event types

The recognized event types are:

- `Caravan`
- `Migrants`
- `Diplomat`
- `FeatureAttack` (Underworld incursion)
- `Megabeast`
- `WildlifeCurious`
- `WildlifeMischievous`
- `WildlifeFlier`
- `NightCreature`

4.6.112 forceequip

Tags: adventure | fort | animals | items | military | units

Command: `forceequip`

Move items into a unit's inventory.

This tool 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. Make sure the unit you want to equip is standing on the target items, which must be on the ground and be unforbidden. If multiple units are standing on the same tile, the first one will be equipped.

The most reliable way to set up the environment for this command is to pile target items on a tile of floor with a garbage dump activity zone or the *autodump* command, then walk/pasture a unit (or use *gui/teleport*) on top of the items. Be sure to unforbid the items that you want to work with!

Note: Weapons are not currently supported.

Usage

```
forceequip [<options>]
```

As mentioned above, this plugin can be used to equip items onto units (such as animals) who cannot normally equip gear. There's an important caveat here: such creatures will automatically drop inappropriate gear almost immediately (within 10 game ticks). If you want them to retain their equipment, you must forbid it AFTER using forceequip to get it into their inventory. This technique can also be used to clothe dwarven infants, but only if you're able to separate them from their mothers.

By default, the `forceequip` command will attempt to abide by game rules as closely as possible. For instance, it will skip any item which is flagged for use in a job, and will not equip more than one piece of clothing/armor onto any given body part. These restrictions can be overridden via options, but doing so puts you at greater risk of unexpected consequences. For instance, a dwarf who is wearing three breastplates will not be able to move very quickly.

Items equipped by this plugin DO NOT become owned by the recipient. Adult dwarves are free to adjust their own wardrobe, and may promptly decide to doff your gear in favour of their owned items. Animals, as described above, will tend to discard ALL clothing immediately unless it is manually forbidden. Armor items seem to be an exception: an animal will tend to retain an equipped suit of mail even if you neglect to forbid it.

Please note that armored animals are quite vulnerable to ranged attacks. Unlike dwarves, animals cannot block, dodge, or deflect arrows, and they are slowed by the weight of their armor.

Examples

`forceequip` Attempts to equip all of the clothing and armor under the cursor onto the unit under the cursor, following game rules regarding which item can be equipped on which body part and only equipping 1 item onto each body part. Items owned by other dwarves are ignored.

`forceequip v bp QQQ` List the bodyparts of the selected unit.

`forceequip bp LH` Equips an appropriate item onto the unit's left hand.

`forceequip m bp LH` Equips ALL appropriate items onto the unit's left hand. The unit may end up wearing a dozen left-handed mittens. Use with caution, and remember that dwarves tend to drop extra items ASAP.

`forceequip i bp NECK` Equips an item around the unit's neck, ignoring appropriateness restrictions. If there's a millstone or an albatross carcass sitting on the same square as the targeted unit, then there's a good chance that it will end up around his neck. For precise control, remember that you can selectively forbid some of the items that are piled on the ground.

`forceequip s` Equips the item currently selected in the k menu, if possible.

`forceequip s m i bp HD` Equips the selected item onto the unit's head. Ignores all restrictions and conflicts. If you know exactly what you want to equip, and exactly where you want it to go, then this is the most straightforward and reliable option.

Options

`i, ignore` Bypasses the usual item eligibility checks (such as "Never equip gear belonging to another dwarf" and "Nobody is allowed to equip a Hive".

`m, multi` Bypasses the 1-item-per-bodypart limit. Useful for equipping both a mitten and a gauntlet on the same hand (or twelve breastplates on the upper body).

`m2, m3, m4` Modifies the 1-item-per-bodypart limit, allowing each part to receive 2, 3, or 4 pieces of gear.

`s, selected` Equip only the item currently selected in the k menu and ignore all other items in the tile.

`bp, bodypart <body part code>` Specify which body part should be equipped.

`v, verbose` Provide detailed narration and error messages, including listing available body parts when an invalid `bodypart` code is specified.

4.6.113 forget-dead-body

Tags: fort | armok | units

Command: `forget-dead-body`

Removes emotions associated with seeing a dead body.

This tool can help your dwarves recover from seeing a massacre. It removes all emotions associated with seeing a dead body. If your dwarves are traumatized and despondent after seeing a dead body, this tool can help.

Usage

`forget-dead-body` Make the selected unit forget seeing dead bodies.

`forget-dead-body --all` Make all units forget seeing dead bodies.

4.6.114 forum-dwarves

Tags: dfhack

Command: `forum-dwarves`

Exports the text you see on the screen for posting to the forums.

Keybinding: CtrlShiftF in `textviewer`

This tool saves a copy of a text screen, formatted in BBcode for posting to the Bay12 Forums. Text color and layout is preserved. See *markdown* if you want to export for posting to Reddit or other places.

This script will attempt to read the current 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 `forumdwarves.txt` file. Previous entries in the file are not overwritten, so you may use the `forum-dwarves` command multiple times to create a single document containing the text from multiple screens, like thoughts from several dwarves or descriptions from multiple artifacts.

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. An error message will inform you when the selected screen is not appropriate for this script.

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 (e.g. *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 copying the text.

Usage

```
forum-dwarves
```

4.6.115 fpause

Tags: dfhack

Command: `fpause`

Forces DF to pause.

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

Usage

```
fpause
```

4.6.116 full-heal

Tags: fort | armok | units

Command: `full-heal`

Fully heal the selected unit.

This script attempts to heal the selected unit from anything, optionally including death.

Usage

`full-heal` Completely heal the currently selected unit.

`full-heal --unit <unitId>` Completely heal the unit with the given ID.

`full-heal -r [--keep_corpse]` Heal the unit, raising from the dead if needed. If `--keep_corpse` is specified, don't remove their corpse. The unit can be targeted by selecting its corpse in the UI.

`full-heal --all [-r] [--keep_corpse]` Heal all units on the map, optionally resurrecting them if dead.

`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 civilization, including pets and visitors.

Examples

`full-heal` Fully heal the selected unit.

`full-heal -r --keep_corpse --unit 23273` Fully heal unit 23273. If this unit was dead, it will be resurrected without removing the corpse - creepy!

4.6.117 gaydar

Tags: fort | inspection | animals | units

Command: `gaydar`

Shows the sexual orientation of units.

`gaydar` is useful for social engineering or checking the viability of livestock breeding programs.

Usage

```
gaydar [<target>] [<filter>]
```

Examples

`gaydar` Show sexual orientation of the selected unit.

`gaydar --citizens --asexual` Identify asexual citizens.

Target options

`--all` Selects every creature on the map.

`--citizens` Selects fort citizens.

`--named` Selects all named units on the map.

Filter options

`--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.

4.6.118 geld

Tags: fort | armok | animals

Command: `geld`

Geld and ungeld animals.

Usage

```
geld [--ungeld|--toggle] [--unit <id>]
```

Examples

geld Gelds the selected animal.

geld --toggle Toggles the gelded status for the selected animal.

geld --ungeld --unit 24242 Ungelds the unit with the specified id.

Options

--unit <id> Selects the unit with the specified ID.

--ungeld Ungelds the specified unit instead of gelding it (see also *ungeld*).

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

4.6.119 generated-creature-renamer

Tags: adventure | fort | legends | units

Automatically renames generated creatures.

Command: `list-generated`

List the token names of all generated creatures.

Command: `save-generated-raws`

Export a creature graphics file for modding.

Now, forgotten beasts, titans, necromancer experiments, etc. will have raw token names that match the description given in-game instead of unreadable generated strings.

Usage

`enable generated-creature-renamer` Rename generated creatures when a world is loaded.

`list-generated [detailed]` List the token names of all generated creatures in the loaded save. If `detailed` is specified, then also show the accompanying description.

`save-generated-raws` 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 world.

4.6.120 getplants

Tags: fort | productivity | plants

Command: `getplants`

Designate trees for chopping and shrubs for gathering.

Specify the types of trees to cut down and/or shrubs to gather by their plant names.

Usage

`getplants [-t|-s|-f]` List valid tree/shrub ids, optionally restricted to the specified type.

`getplants <id> [<id> ...] [<options>]` Designate trees/shrubs of the specified types for chopping/gathering.

Examples

`getplants` List all valid IDs.

`getplants -f -a` Gather all plants on the map that yield seeds for farming.

`getplants NETHER_CAP -n 10` Designate 10 nether cap trees for chopping.

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.

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

`-c` Clear: Clear designations instead of setting them.

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

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

`-n <num>` Number: Designate up to the specified number of plants of each species.

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.

4.6.121 ghostly

Tags: adventure | armok | units

Command: `ghostly`

Toggles an adventurer's ghost status.

This is useful for walking through walls, avoiding attacks, or recovering after a death.

Usage

```
ghostly
```

4.6.122 growcrops

Tags: fort | armok | plants

Command: `growcrops`

Instantly grow planted seeds into crops.

With no parameters, this command lists the seed types currently planted in your farming plots. With a seed type, the script will grow those seeds, ready to be harvested.

Usage

`growcrops` List the seeds planted in your farming plots.

`growcrops <crop name>` Grow the specified planted seed type. The given name can be a substring of the full crop name.

`growcrops all` Grow all planted seeds.

Example

growcrops plump Grow all planted plump helmet spawn seeds.

4.6.123 help

Tags: dfhack

Command: `help`

Display help about a command or plugin.

Can also be invoked as `?` or `man` (short for "manual").

Usage

```
help|?|man
help|?|man <command or plugin>
```

Examples

```
help blueprint
man blueprint
```

Both examples above will display the help text for the *blueprint* command.

Some commands also take `help` or `?` as an option on their command line for the same effect – e.g. `blueprint help`.

4.6.124 hermit

Tags: fort | gameplay

Command: `hermit`

Go it alone in your fortress and attempt the hermit challenge.

This script blocks all caravans, migrants, diplomats, and forgotten beasts (not wildlife) from entering your fort. Useful for attempting the hermit challenge.

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

Usage

```
enable hermit
```

4.6.125 hfs-pit

Tags: fort | armok | map

Command: `hfs-pit`

Creates a pit straight to the underworld.

This script creates a pit to the underworld, starting at the cursor position and going down, down down.

Usage

```
hfs-pit [<size> [<walls> [<stairs>]]]
```

The first parameter is the "radius" in tiles of the (square) pit, that is, how many tiles to open up in each direction around the cursor. The default is 1, meaning a single column.

The second parameter 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 in the middle of the pit or anything else to just have an open channel.

Note that stairs are buggy; they will not reveal the bottom until you dig somewhere, but underworld creatures will path in.

Examples

`hfs-pit` Create a single-tile wide pit with no walls or stairs.

`hfs-pit 4 1 0` A seven-across pit (the center tile plus three on each side) with stairs but no containing walls.

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

4.6.126 hide

Tags: dfhack

Command: `hide`

Hide the DFHack terminal window.

You can show it again with the *show* command, though you'll need to use it from a *keybinding* set beforehand or the in-game *command-prompt*.

Only available on Windows.

Usage

```
hide
```

4.6.127 hotkey-notes

Tags: fort | inspection

Command: `hotkey-notes`

Show info on DF map location hotkeys.

This command lists the key (e.g. `F1`), name, and jump position of the map location hotkeys you set in the `H` menu.

Usage

```
hotkey-notes
```

4.6.128 hotkeys

Tags: dfhack

Command: `hotkeys`

Show all DFHack keybindings for the current context.

Keybinding: CtrlShiftC

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

Usage

`hotkeys` Show the list of keybindings for the current context in an in-game menu.

`hotkeys list` List the keybindings to the console.

Menu overlay widget

The in-game hotkeys menu is registered with the *overlay* framework and can be enabled as a hotspot in the upper-left corner of the screen. You can bring up the menu by hovering the mouse cursor over the hotspot and can select a command to run from the list by clicking on it with the mouse or by using the keyboard to select a command with the arrow keys and hitting `Enter`.

A short description of the command will appear in a nearby textbox. If you'd like to see the full help text for the command or edit the command before running, you can open it for editing in *gui/launcher* by right clicking on the command, left clicking on the arrow to the left of the command, or by pressing the right arrow key while the command is selected.

4.6.129 infiniteSky

Tags: fort | design | map

Command: `infiniteSky`

Automatically allocate new z-levels of sky

If enabled, this plugin will automatically allocate new z-levels of sky at the top of the map as you build up. Or it can allocate one or many additional levels at your command.

Usage

`enable infiniteSky` Enables monitoring of constructions. If you build anything in the second highest z-level, it will allocate one more sky level. You can build stairs up as high as you like!

`infiniteSky [<n>]` Raise the sky by n z-levels. If run without parameters, raises the sky by one z-level.

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

4.6.130 install-info

Tags: dfhack

Command: `install-info`

Exports information about DFHack for bug reports.

This command saves information about the current DFHack installation and recent errors to `install-info.txt` in the current DF folder. Useful for bug reports.

Usage

```
install-info
```

4.6.131 isoworldremote

Tags: dev | graphics

Provides a remote API used by Isoworld.

See *DFHack remote interface* for related remote APIs.

4.6.132 jobutils

Tags: fort | inspection | jobs

Provides commands for interacting with jobs.

Command: `job`

Inspect or modify details of workshop jobs.

Command: `job-duplicate`

Duplicates the highlighted job.

Keybinding: CtrlD

Command: `job-material`

Alters the material of the selected job.

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"`

Usage

`job` Print details of the current job. The job can be selected in a workshop or the unit/jobs screen.

`job list` Print details of all jobs in the selected workshop.

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

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

`job-duplicate` Duplicates the highlighted job. Must be in `q` mode and have a workshop or furnace building selected.

`job-material <inorganic-token>` Alters the material of the selected job (in `q` mode) or jumps to the selected material when choosing the building component of a planned building (in `b` mode). Note that this form of the command can only handle inorganic materials.

Use the `job` and `job list` commands to discover the type and material ids for existing jobs, or use the following commands to see the full lists:

```
lua @df.item_type
lua "for i,mat in ipairs(df.global.world.raws.inorganics) do if mat.material.flags.IS_
↪STONE and not mat.material.flags.NO_STONE_STOCKPILE then print(i, mat.id) end end"
```

Examples

`job-material GNEISS` Change the selected "Construct rock Coffin" job at a Mason's workshop to "Construct gneiss coffin".

`job item-material 2 MARBLE` Change the selected "Construct Traction Bench" job (which has three source items: a table, a mechanism, and a chain) to specifically use a marble mechanism.

`job item-type 2 TABLE` Change the selected "Encrust furniture with blue jade" job (which has two source items: a cut gem and a piece of improvable furniture) to specifically use a table instead of just any furniture.

4.6.133 keybinding

Tags: dfhack

Command: `keybinding`

Create hotkeys that will run DFHack commands.

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-config/init/dfhack.init`.

Hotkeys can be any combinations of Ctrl/Alt/Shift with A-Z, 0-9, F1-F12, or ` (the key below the `Esc` key on most keyboards).

Usage

`keybinding` Show some useful information, including the current game context.

`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.

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`.

Commands like *liquids* or *tiletypes* cannot be used as hotkeys since they require the console for interactive input.

Examples

`keybinding add Ctrl-Shift-C hotkeys` Bind Ctrl-Shift-C to run the *hotkeys* command on any screen at any time.

`keybinding add Alt-F@dwarfmode gui/quickfort` Bind Alt-F to run *gui/quickfort*, but only when on a screen that shows the main map.

`keybinding add Ctrl-Shift-Z@dwarfmode/Default "stocks show"` Bind Ctrl-Shift-Z to run *stocks show*, but only when on the main map in the default mode (that is, no special mode, like cursor look, is enabled).

4.6.134 kill-lua

Tags: dfhack

Command: `kill-lua`

Gracefully stop any currently-running Lua scripts.

Use this command to stop a misbehaving script that appears to be stuck.

Usage

```
kill-lua
kill-lua force
```

Use `kill-lua force` if just `kill-lua` doesn't seem to work.

4.6.135 labormanager

Tags: fort | auto | labors

Command: `labormanager`

Automatically manage dwarf labors.

Labormanager is derived from *autolabor* 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 dwarves, 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.

Dwarves on active military duty or dwarves assigned to burrows are left untouched.

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. Do not run both autolabor and labormanager at the same time!

Usage

```
enable labormanager
```

Anything beyond this is optional - labormanager works well with the default settings. Once you have enabled it in a fortress, it stays enabled until you explicitly disable it, even if you save and reload your game.

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 jobs of each labor are 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 dwarves 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 dwarves 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".

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 stockpiles to haul food to.

Dwarves 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 dwarves generally attempts to favor dwarves 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 dwarves 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 dwarves 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 (see below).

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 <issue> on GitHub, reporting the lines that immediately preceded this message. You can tell labormanager to ignore this error and carry on by running `labormanager pause-on-error no`, but be warned that some job may go undone in this situation.

Examples

`labormanager priority BREWER 500` Boost the priority of brewing jobs.

`labormanager max FISH 1` Only assign fishing to one dwarf at a time. Note that you also have to run `labormanager allow-fishing` for any dwarves to be assigned fishing at all.

Advanced usage

`labormanager list` Show current priorities and current allocation stats. Use this command to see the IDs for all labors.

`labormanager status` Show basic status information.

`labormanager priority <labor> <value>` Set the priority value for labor <labor> to <value>.

`labormanager max <labor> <value>` Set the maximum number of dwarves that can be assigned to a labor.

`labormanager max <labor> none` Unrestrict the number of dwarves that can be assigned to a labor.

`labormanager max <labor> disable` Don't manage the specified labor. Dwarves who you have manually enabled this labor on will be less likely to have managed labors assigned to them.

`labormanager reset-all|reset <labor>` Return a labor (or all labors) to the default priority.

`labormanager allow-fishing|forbid-fishing` Allow/disallow fisherdwarves. *Warning* Allowing fishing tends to result in most of the fort going fishing. Fishing is forbidden by default.

`labormanager allow-hunting|forbid-hunting` Allow/disallow hunterdwarves. *Warning* Allowing hunting tends to result in as many dwarves going hunting as you have crossbows. Hunting is forbidden by default.

`labormanager pause-on-error yes|no` Make labormanager pause/continue if the labor inference engine fails. See the above section for details.

4.6.136 lair

Tags: fort | armok | map

Command: `lair`

Mark the map as a monster lair.

This avoids item scatter when the fortress is abandoned.

Usage

`lair` Mark the map as a monster lair.

`lair reset` Mark the map as ordinary (not lair).

This command doesn't save the information about tiles - you won't be able to restore the state of a real monster lairs using `lair reset`.

4.6.137 launch

Tags: adventure | armok | units

Command: `launch`

Thrash your enemies with a flying suplex.

Attack another unit and then run this command to grab them and fly in a glorious parabolic arc to where you have placed the cursor. You'll land safely and your opponent will slam into the ground, skidding away. Extra points for skidding them into a wall or off a cliff! The farther you jump, the harder they slam. Launch six tiles away or more for some serious pain!

Note that if you run this command *without* attacking an enemy first, it will be *you* that slams into the ground. Launching yourself 3 tiles or so should be fine (e.g. to surprise an approaching enemy), but don't expect to survive after jumping across the map!

Usage

```
launch
```

4.6.138 lever

Tags: fort | armok | inspection | productivity | buildings

Command: `lever`

Inspect and pull levers.

Usage

`lever list` Print out a list of your fort's levers, including their name, activation state, and what they are linked to (if anything).

`lever pull <id> [<options>]` Queue a job so a dwarf will pull the specified lever. This is the same as `q` querying the building and queueing a `P` pull job.

If your levers aren't named, you can find out a lever's ID with the *bprobe* command.

Examples

`lever pull 42 --priority` Queue a job to pull lever 42 at high priority.

`lever pull 42 --now` Skip the job and pull the lever with the hand of Armok!

Options

`--priority` Queue a job at high priority.

`--now` Magically toggle the lever immediately.

4.6.139 light-aquifers-only

Tags: embark | fort | armok | map

Command: `light-aquifers-only`

Change heavy and varied aquifers to light aquifers.

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 modifies the map tiles, leaving the rest of the world unchanged.

Usage

```
light-aquifers-only
```

If you don't ever want to have to deal with heavy aquifers, you can add the `light-aquifers-only` command to your `dfhack-config/init/onMapLoad.init` file.

Technical details

When run pre-embark, this script changes the drainage of all world tiles that would generate Heavy aquifers into a value that results in Light aquifers instead, 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.

When run post-embark, this script simply clears the flags that mark aquifer tiles as heavy, converting them to light.

4.6.140 linger

Tags: adventure | armok

Command: `linger`

Take control of your adventurer's killer.

Run this script after being presented with the "You are deceased." message to abandon your dead adventurer and take control of your adventurer's killer.

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.

Usage

```
linger
```

4.6.141 liquids

Tags: adventure | fort | armok | map

Command: `liquids`

Place magma, water or obsidian.

Command: `liquids-here`

Spawn liquids on the selected tile.

Place magma, water or obsidian. See *gui/liquids* for an in-game interface for this functionality.

Also, if you only want to add or remove water or magma from a single tile, the *source* script may be easier to use.

Usage

`liquids` Start the interactive terminal settings interpreter. This command must be called from the DFHack terminal and not from any in-game interface.

`liquids-here` Run the liquid spawner with the current/last settings made in `liquids` (if no settings in `liquids` were made, then it paints a point of 7/7 magma by default). This command is intended to be used as keybinding, and it requires an active in-game cursor.

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

Interactive interpreter

The interpreter replaces the normal dfhack command line and can't be used from a hotkey. Settings will be remembered as long as dfhack runs. It is intended for use in combination with the command `liquids-here` (which *can* be bound to a hotkey).

You can enter the following commands at the prompt.

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)

4.6.142 list-agreements

Tags: fort | inspection

Command: `list-agreements`

List guildhall and temple agreements.

If you have trouble remembering which guildhalls and temples you've agreed to build (and don't we all?), then this script is for you! You can use this command to clearly list outstanding agreements or see the entire history of agreements for your fort.

Usage

```
list-agreements [all]
```

Examples

`list-agreements` List outstanding, unfulfilled location agreements.

`list-agreements all` Lists all location agreements, whether satisfied, denied, or expired.

4.6.143 list-waves

Tags: fort | inspection | units

Command: `list-waves`

Show migration wave information for your dwarves.

This script displays information about migration waves or identifies which wave a particular dwarf came from.

Usage

```
list-waves --all [--showarrival] [--granularity <value>]
list-waves --unit [--granularity <value>]
```

Examples

`list-waves --all` Show how many dwarves came in each migration wave.

`list-waves --all --showarrival` Show how many dwarves came in each migration wave and when that migration wave arrived.

`list-waves --unit` Show which migration wave the selected dwarf arrived with.

Options

`--unit` Displays the highlighted unit's arrival wave information.

`--all` Displays information about each arrival wave.

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

`--showarrival:` Shows the arrival date for each wave.

4.6.144 load

Tags: dfhack

Command: `load`

Load and register a plugin library.

Also see *unload* and *reload* for related actions.

Usage

```
load <plugin> [<plugin> ...]
load -a|--all
```

You can load individual named plugins or all plugins at once. Note that plugins are disabled after loading/reloading until you explicitly *enable* them.

4.6.145 load-save

Tags: dfhack

Command: `load-save`

Load a savegame.

When run on the Dwarf Fortress title screen or "load game" screen, this script will load the save with the given folder name without requiring interaction. Note that inactive saves (i.e. saves under the "start game" menu that have gone through world generation but have not had a fort or adventure game started in them yet) cannot be loaded by this script.

Usage

```
load-save <save directory name>
```

Examples

`load-save region1` Load the savegame in the `data/save/region1` directory.

Autoloading a game on DF start

It is useful to run this script from the commandline when starting Dwarf Fortress. For example, on Linux/MacOS you could start Dwarf Fortress with:

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

Similarly, on Windows, you could run:

```
"Dwarf Fortress.exe" +load-save region1
```

4.6.146 locate-ore

Tags: fort | armok | productivity | map

Command: `locate-ore`

Scan the map for metal ores.

This tool finds and designates for digging one tile of a specific metal ore. If you want to dig **all** tiles of that kind of ore, select that tile with the cursor and run *digtype*.

Usage

`locate-ore list` List metal ores available on the map.

`locate-ore <type>` Finds a tile of the specified ore type, zooms the screen so that tile is visible, and designates that tile for digging.

Examples

```
locate-ore hematite
locate-ore iron
locate-ore silver
```

Note that looking for a particular metal might find an ore that contains that metal along with other metals. For example, locating silver may find tetrahedrite, which contains silver and copper.

4.6.147 ls

Tags: dfhack

Command: `ls`

List available DFHack commands.

In order to group related commands, each command is associated with a list of tags. You can filter the listed commands by a tag or a substring of the command name. Can also be invoked as `dir`.

Usage

`ls [<options>]` Lists all available commands and the tags associated with them.

`ls <tag> [<options>]` Shows only commands that have the given tag. Use the *tags* command to see the list of available tags.

`ls <string> [<options>]` Shows commands that include the given string. E.g. `ls quick` will show all the commands with “quick” in their names. If the string is also the name of a tag, then it will be interpreted as a tag name.

Examples

`ls quick` List all commands that match the substring “quick”.

`ls adventure` List all commands with the `adventure` tag.

`ls --dev trigger` List all commands, including developer and modding commands, that match the substring “trigger”.

Options

`--notags` Don't print out the tags associated with each command.

`--dev` Include commands intended for developers and modders.

`--exclude <string>[,<string>...]` Exclude commands that match any of the given strings.

4.6.148 lua

Tags: dev

Command: `lua`

Run Lua script commands.

Usage

`lua` Start an interactive lua interpreter.

`lua -f <filename>`, `lua --file <filename>` Load the specified file and run the lua script within. The filename is interpreted relative to the Dwarf Fortress game directory.

`lua -s [<filename>]`, `lua --save [<filename>]` Load the specified file and run the lua script within. The filename is interpreted relative to the current save directory. If the filename is not supplied, it loads `dfhack.lua`.

`:lua <lua statement>` Parses and executes the given lua statement like the interactive interpreter would.

The last form recognizes shortcut characters from the interactive interpreter for easy inspection of values:

```
'! foo' => 'print(foo)'
'~ foo' => 'printall(foo)'
'^ foo' => 'printall_recurse(foo)'
'@ foo' => 'printall_ipairs(foo)'
```

Examples

`:lua !df.global.window_z` Print out the current z-level.

4.6.149 luasocket

Tags: dev

Provides a Lua API for accessing network sockets.

See *luasocket* for details.

4.6.150 make-legendary

Tags: fort | armok | units

Command: `make-legendary`

Boost skills of the selected dwarf.

This tool can make the selected dwarf legendary in one skill, a group of skills, or all skills.

Usage

`make-legendary list` List the individual skills that you can boost.

`make-legendary classes` List the skill groups that you can boost.

`make-legendary <skill>|all` Make the selected dwarf legendary in the specified skill (or all skills)

Examples

`make-legendary MINING` Make the selected dwarf a legendary miner.

`make-legendary all` Make the selected dwarf legendary in all skills. Only perfection will do.

4.6.151 make-monarch

Tags: fort | armok | units

Command: `make-monarch`

Crown the selected unit as a monarch.

This tool can make the selected unit king or queen of your civilization.

Usage

```
make-monarch
```

4.6.152 manipulator

Tags: fort | productivity | labors

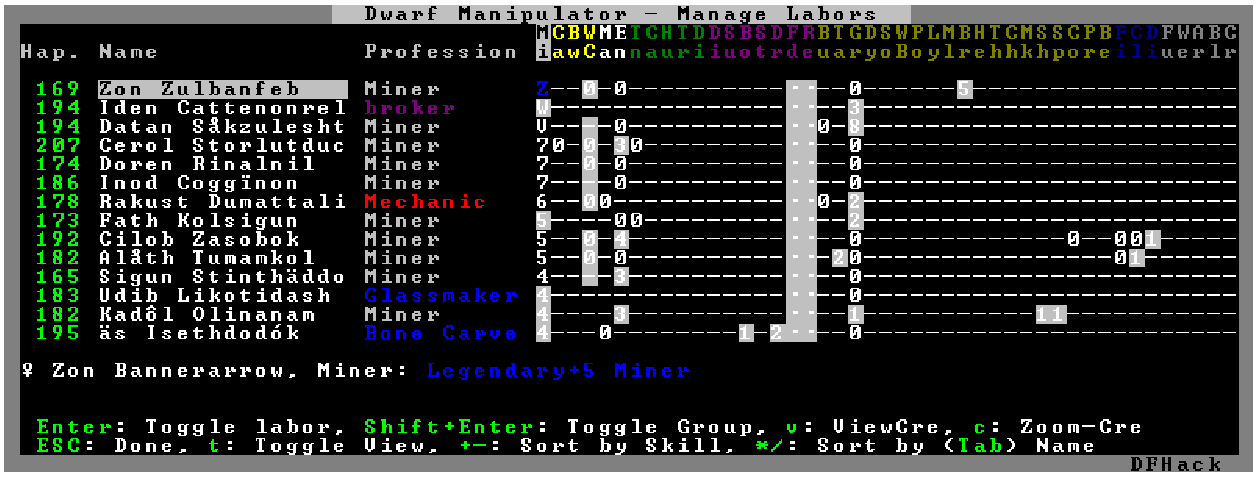
An in-game labor management interface.

It is equivalent to the popular Dwarf Therapist utility.

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

Usage

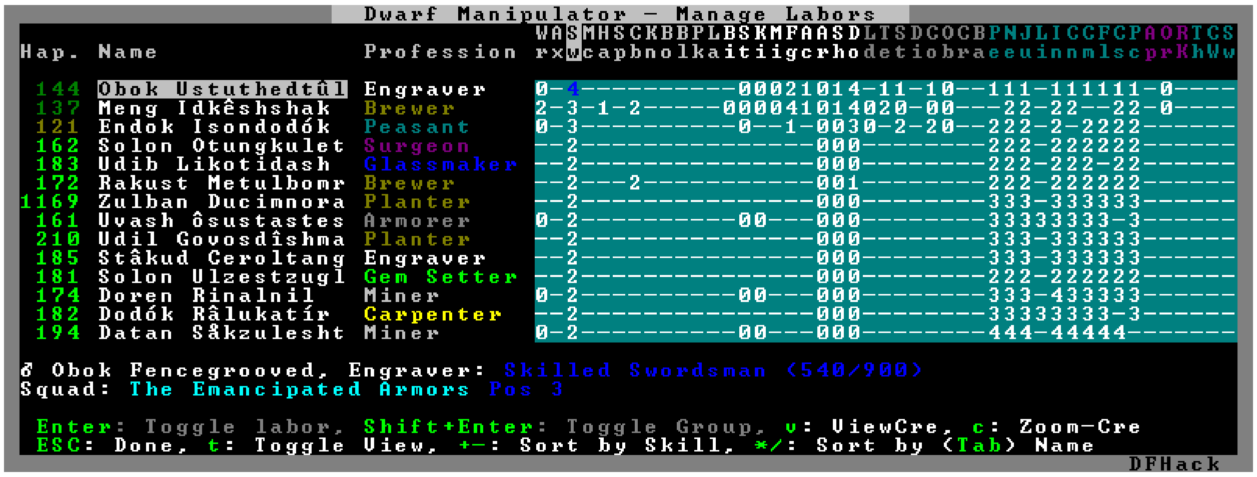
```
enable manipulator
```



```
                           Dwarf Manipulator - Manage Labors
                                          MCBWMETCHTDDSBSDFRBTGDSWPLMBHTCMSSCPBFCDFWABC
Hap. Name               Profession        iawCannauriiuotrdeuaryoBoylrehhkhporeiliuerlr

 169 Zon Zulbanfeb      Miner             Z--0-0----------  --0------5-----------------
 194 Iden Cattenonrel   broker            W---------------  --3------------------------
 194 Datan Såkzulesht   Miner             U-- -0----------  0-8------------------------
 207 Cerol Storlutduc   Miner             70-0-30---------  --0------------------------
 174 Doren Rinalnil     Miner             7--0-0----------  --0------------------------
 186 Inod Coggïnon      Miner             7-- -0----------  --0------------------------
 178 Rakust Dumattali   Mechanic          6--00-----------  0-2------------------------
 173 Fath Kolsigun      Miner             5----00---------  --2------------------------
 192 Cilob Zasobok      Miner             5--0-4----------  --0--------------0--001----
 182 Alåth Tumamkol     Miner             5--0-0----------  -20-----------------01-----
 165 Sigun Stinthäddo   Miner             4-- -3----------  --0------------------------
 183 Udib Likotidash    Glassmaker        4---------------  --0------------------------
 182 Kadôl Olinanam     Miner             4----3----------  --1------------11----------
 195 äs Isethdodók      Bone Carve        4---0--------1-2  --0------------------------

♀ Zon Bannerarrow, Miner: Legendary+5 Miner

 Enter: Toggle labor, Shift+Enter: Toggle Group, v: ViewCre, c: Zoom-Cre
 ESC: Done, t: Toggle View, +-: Sort by Skill, */: Sort by (Tab) Name
                                                                      DFHack
```

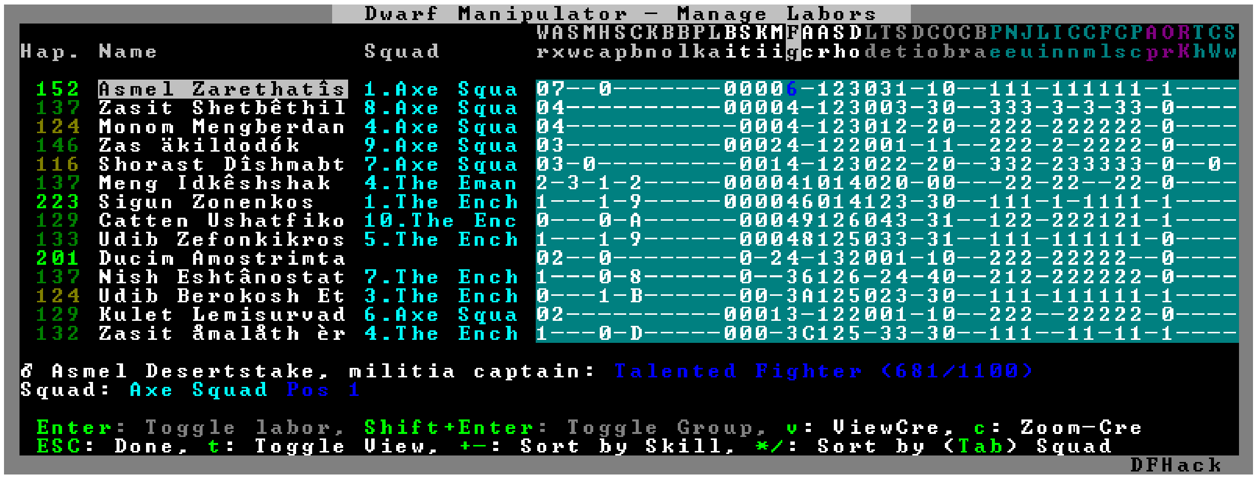
The far left column displays the unit's name, happiness (color-coded based on its value), profession, or squad, and the right half of the screen displays each dwarf's labor settings and skill levels (0-9 for Dabbling through Professional, A-E for Great through Grand Master, and U-Z for Legendary through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

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



```
                           Dwarf Manipulator - Manage Labors
                                          WASMHSCKBBPLBSKMFAASDLTSDCOCBPNJLICCFCPAORTCS
Hap. Name               Profession        rxwcapbnolkaitiigcrhodetiobraeeuinnmlscprKhWw

 144 Obok Ustuthedtûl   Engraver          0-4----------00021014-11-10--111-111111-0----
 137 Meng Idkêshshak    Brewer            2-3-1-2-----000041014020-00---22-22---22-0---
 121 Endok Isondodók    Peasant           0-3----------0--1-0030-2-20--222-2-2222------
 162 Solon Otungkulet   Surgeon           --2---------------000--------222-222222------
 183 Udib Likotidash    Glassmaker        --2---------------000--------222-222-22------
 172 Rakust Metulbomr   Brewer            --2---2-----------001--------222-222222------
1169 Zulban Ducimnora   Planter           --2---------------000--------333-333333------
 161 Uvash ôsustastes   Armorer           0-2----------00---000--------3333333-3-------
 210 Udil Govosdîshma   Planter           --2---------------000--------333-333333------
 185 Stâkud Ceroltang   Engraver          --2---------------000--------333-333333------
 181 Solon Ulzestzugl   Gem Setter        --2---------------000--------222-222222------
 174 Doren Rinalnil     Miner             0-2----------00---000--------333-433333------
 182 Dodók Râlukatír    Carpenter         --2---------------000--------3333333-3-------
 194 Datan Såkzulesht   Miner             0-2----------00---000--------444-44444-------

♂ Obok Fencegrooved, Engraver: Skilled Swordsman (540/900)
Squad: The Emancipated Armors Pos 3

 Enter: Toggle labor, Shift+Enter: Toggle Group, v: ViewCre, c: Zoom-Cre
 ESC: Done, t: Toggle View, +-: Sort by Skill, */: Sort by (Tab) Name
                                                                      DFHack
```

Press `t` to toggle between Profession, Squad, and Job views.



```
                                Dwarf Manipulator - Manage Labors
                                          WASMHSCKBBPLBSKMFAASDLTSDCOCBPNJLICCFCPAORTCS
Hap. Name               Squad             rxwcapbnolkaitiigcrhodetiobraeeuinnmlscprKhWw

 152 Asmel Zarethatîs   1.Axe Squa        07--0-------00006-123031-10--111-111111-1----
 137 Zasit Shetbêthil   8.Axe Squa        04----------00004-123003-30--333-3-3-33-0----
 124 Monom Mengberdan   4.Axe Squa        04-----------0004-123012-20--222-222222-0----
 146 Zas äkildodók      9.Axe Squa        03----------00024-122001-11--222-2-2222-0----
 116 Shorast Dîshmabt   7.Axe Squa        03-0---------0014-123022-20--332-233333-0--0-
 137 Meng Idkêshshak    4.The Eman        2-3-1-2-----000041014020-00---22-22--22-0----
 223 Sigun Zonenkos     1.The Ench        1---1-9-----000046014123-30--111-1-1111-1----
 129 Catten Ushatfiko   10.The Enc        0---0-A------00049126043-31--122-222121-1----
 133 Udib Zefonkikros   5.The Ench        1---1-9------00048125033-31--111-111111-0----
 201 Ducim Amostrimta                     02--0--------0-24-132001-10--222-22222--0----
 137 Nish Eshtânostat   7.The Ench        1---0-8------0--36126-24-40--212-222222-0----
 124 Udib Berokosh Et   3.The Ench        0---1-B------00-3A125023-30--111-111111-1----
 129 Kulet Lemisurvad   6.Axe Squa        02----------00013-122001-10--222--22222-0----
 132 Zasit åmalåth èr   4.The Ench        1---0-D-----000-3C125-33-30--111--11-11-1----

♂ Asmel Desertstake, militia captain: Talented Fighter (681/1100)
Squad: Axe Squad Pos 1

 Enter: Toggle labor, Shift+Enter: Toggle Group, v: ViewCre, c: Zoom-Cre
 ESC: Done, t: Toggle View, +-: Sort by Skill, */: Sort by (Tab) Squad
                                                                          DFHack
```

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 profession name 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 and the custom profession names for those dwarves will be set to the applied profession.

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

The professions library

The manipulator plugin comes with a library of professions that you can assign to your dwarves.

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 list below, the "needed" range indicates 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. These are just approximations. Your playstyle may demand more or fewer of each profession.

- **`Chef` (needed: 0, 3)** Butchery, 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` (needed: 0, 4-6)** All labors used at Craftsdwarf's workshops, Glassmaker's workshops, and kilns.
- **`Doctor` (needed: 0, 2-4)** The full suite of medical labors, plus Animal Caretaking for those using the *dwarfvet* plugin.
- **`Farmer` (needed 1, 4)** Food- and animal product-related labors.
- **`Fisherdwarf` (needed 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 else caught fish will just be left to rot.
- **`Hauler` (needed 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 unskilled haulers don't make low-quality mechanisms.
- **`Laborer` (needed 0, 10-12)** All labors that don't improve quality with skill, such as Soapmaking and furnace labors.
- **`Marksdwarf` (needed 0, 10-30)** Similar to `Hauler`. See the description for `Meleedwarf` below for more details.
- **`Mason` (needed 2, 2-4)** Masonry and Gem Cutting/Encrusting.
- **`Meleedwarf` (needed 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 and reset traps.
- **`Migrant` (needed 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` (needed 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` (needed 1, 2-4)** Carpentry, Bowyery, Woodcutting, Animal Training, Trapping, Plant Gathering, Beekeeping, and Siege Engineering.
- **`Smith` (needed 0, 2-4)** Smithing labors. You may want to specialize your Smiths to focus on a single smithing skill to maximize equipment quality.
- **`StartManager` (needed 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` (needed 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
```

4.6.153 map-render

Tags: dev | graphics

Provides a Lua API for re-rendering portions of the map.

See *map-render* for details.

4.6.154 markdown

Tags: dfhack

Command: `markdown`

Exports the text you see on the screen for posting online.

This tool saves a copy of a text screen, formatted in markdown, for posting to Reddit (among other places). See *forum-dwarves* if you want to export BBCode for posting to the Bay 12 forums.

This script will attempt to read the current screen, and if it is a text viewscreen (such as the dwarf 'thoughts' screen or an item 'description') then append a marked-down version of this text to the output file. 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, like thoughts from several dwarves or descriptions from multiple artifacts.

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 pages
5. announcements 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. An error message will inform you when the selected screen is not appropriate for this script.

Usage

```
markdown [-n] [<name>]
```

The output is appended to the `md_export.md` file by default. If an alternate name is specified, then a file named like `md_{name}.md` is used instead.

Examples

`markdown` Appends the contents of the current screen to the `md_export.md` file.

`markdown artifacts` Appends the contents of the current screen to the `md_artifacts.md` file.

Options

`-n` Overwrite the contents of output file instead of appending.

4.6.155 masspit

Tags: fort | productivity | animals

Command: `masspit`

Designate creatures for pitting.

If you have prepared an animal stockpile on top of a pit zone, and that stockpile has been filled with animals/prisoners in cages, then this tool can designate the inhabitants of all those cages for pitting.

Usage

```
masspit [<zone id>]
```

If no zone id is given, use the zone under the cursor.

Examples

`masspit` Pit all animals within the selected zone.

`masspit 6` Pit all animals within the `Activity Zone #6` zone.

4.6.156 max-wave

Tags: fort | units

Command: `max-wave`

Dynamically limit the next immigration wave.

Limit the number of migrants that can arrive in the next wave by overriding the population cap value from data/init/d_init.txt. Use with the *repeat* command to set a rolling immigration limit. Original credit was for Loci.

If you edit the population caps using *gui/settings-manager* after running this script, your population caps will be reset and you may get more migrants than you expected.

Usage

```
max-wave <wave_size> [max_pop]
```

Examples

```
max-wave 5
repeat -time 1 -timeUnits months -command [ max-wave 10 200 ]
```

The first example ensures the next migration wave has 5 or fewer immigrants. The second example ensures all future seasons have a maximum of 10 immigrants per wave, up to a total population of 200.

4.6.157 migrants-now

Tags: fort | armok | units

Command: `migrants-now`

Trigger a migrant wave.

This tool can trigger an immediate migrant wave. Note that it only works after at least one wave of migrants have arrived naturally, and that no migrants are actually guaranteed to arrive (especially if you've just run this command a few times).

Usage

```
migrants-now
```

4.6.158 misery

Tags: fort | armok | units

Command: `misery`

Increase the intensity of negative dwarven thoughts.

When enabled, negative thoughts that your dwarves have will multiply by the specified factor.

Usage

`enable misery` Start multiplying negative thoughts.

`misery <factor>` Change the multiplicative factor of bad thoughts. The default is `2`.

`misery clear` Clear away negative thoughts added by `misery`.

4.6.159 mode

Tags: armok | dev | gameplay

Command: `mode`

See and change the game mode.

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.

Usage

`mode` Print the current game mode.

`mode set` Enter an interactive commandline menu where you can set the game mode.

Examples

Scenario 1:

- 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.

You just lost a fortress and gained an adventurer.

Scenario 2:

- 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.

4.6.160 mousequery

Tags: fort | productivity | interface

Command: `mousequery`

Adds mouse controls to the DF interface.

Adds mouse controls to the DF interface. For example, with `mousequery` you can click on buildings to configure them, hold the mouse button to draw dig designations, or click and drag to move the map around.

Usage

```
enable mousequery
mousequery [rbutton|track|edge|live] [enable|disable]
mousequery drag [left|right|disable]
mousequery delay [<ms>]
```

rbutton When the right mouse button is clicked, cancel out of menus or scroll the main map if you r-click near an edge.

track Move the cursor with the mouse instead of the cursor keys when you are in build or designation modes.

edge Scroll the map when you move the cursor to a map edge. See `delay` below. If enabled also enables `track`.

delay Set delay in milliseconds for map edge scrolling. Omit the amount to display the current setting.

live Display information in the lower right corner of the screen about the items/building/tile under the cursor, even while unpaused.

Examples

`mousequery rbutton enable` Enable using the right mouse button to cancel out of menus and scroll the map.

`mousequery delay 300` When run after `mousequery edge enable`, sets the edge scrolling delay to 300ms.

4.6.161 multicmd

Tags: dfhack

Command: `multicmd`

Run multiple DFHack commands.

This utility command allows you to specify multiple DFHack commands on a single line.

The string is split around the `;` character(s), and all parts are run sequentially as independent dfhack commands. This is especially useful for hotkeys, where you only have one line to specify what the hotkey will do.

Usage

```
multicmd <command>; <command>[; <command> ...]
```

Example

```
multicmd :lua require('gui.dwarfmode').enterSidebarMode(df.ui_sidebar_mode.
↪DesignateMine); locate-ore IRON; digv; digcircle 16
```

4.6.162 names

Tags: fort | productivity | units

Command: `names`

Rename units or items with the DF name generator.

This tool allows you to rename the selected unit or item (including artifacts) with the native Dwarf Fortress name generation interface.

To modify or remove the first name, press `f` and edit.

Usage

```
names
```

4.6.163 nestboxes

Tags: fort | auto | animals

Protect fertile eggs incubating in a nestbox.

This plugin will automatically scan for and forbid fertile eggs incubating in a nestbox so that dwarves won't come to collect them for eating. The eggs will hatch normally, even when forbidden.

Usage

```
enable nestboxes
```

4.6.164 on-new-fortress

Tags: dfhack

Command: `on-new-fortress`

Run commands when a fortress is first started.

This utility command checks to see if the current fortress has just been created (that is, the "age" of the fortress is 0, which is only true on the first tick after the initial embark).

You can specify multiple commands to run, separated with `;`, similar to *multicmd*. However, if the fortress is not brand new, the commands will not actually run.

Usage

```
on-new-fortress <command>[; <command> ...]
```

Example

You can add commands to your `dfhack-config/init/onMapLoad.init` file that you only want to run when a fortress is first started:

```
on-new-fortress ban-cooking tallow; ban-cooking honey; ban-cooking oil
on-new-fortress 3dveins
on-new-fortress enable autobutcher; autobutcher autowatch
```

4.6.165 once-per-save

Tags: dfhack

Command: `once-per-save`

Run commands only if they haven't been run before in this world.

If you are looking for a way to run commands once when you start a new fortress, you probably want *on-new-fortress*.

This tool is better for commands that you want to run once per world.

You can specify multiple commands to run, separated with `;`, similar to *multicmd*. However, if the command has been run (successfully) before with `once-per-save` in the context of the current savegame, the commands will not actually run.

Usage

`once-per-save [--rerun] <command>[; <command> ...]` Run the specified commands if they haven't been run before. If `--rerun` is specified, run the commands regardless of whether they have been run before.

`once-per-save --reset` Forget which commands have been run before.

4.6.166 open-legends

Tags: adventure | fort | legends

Command: `open-legends`

Open a legends screen from fort or adventure mode.

You can use this tool to open legends mode from a world loaded in fortress or adventure mode. You can browse around, or even run *exportlegends* while you're on the legends screen.

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 **before** running `open-legends`
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 game!)

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.

Usage

```
open-legends [force]
```

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

4.6.167 orders

Tags: fort | productivity | workorders

Command: `orders`

Manage manager orders.

Usage

`orders orders list` Shows the list of previously exported orders, including the orders library.

`orders export <name>` Saves all the current manager orders in a file.

`orders import <name>` Imports the specified manager orders. Note this adds to your current set of manager orders. It will not clear the orders that already exist.

`orders clear` Deletes all manager orders in the current embark.

`orders sort` Sorts current manager orders by repeat frequency so repeating orders don't prevent one-time orders from ever being completed. The sorting order is: one-time orders first, then yearly, seasonally, monthly, and 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 ]
```

Exported orders are saved in the `dfhack-config/orders` directory, where you can view, edit, and delete them, if desired.

Examples

`orders export myorders` Export the current manager orders to a file named `dfhack-config/orders/myorders.json`.

`orders import library/basic` Import manager orders from the library that keep your fort stocked with basic essentials.

The orders library

DFHack comes with a library of useful manager orders that are ready for import:

library/basic

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, totems, and shleggings 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.

Armok's note: shleggings? Yes, shleggings.

library/furnace

This collection creates basic items that require heat. It is separated out from `library/basic` 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.

library/military

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.

library/smelting

This collection adds smelting jobs for all ores. It includes handling the ores already managed by `library/military`, but has lower limits. This ensures all ores will be covered if a player imports `library/smelting` but not `library/military`, but the higher-volume `library/military` orders will take priority if both are imported.

library/rockstock

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.

library/glassstock

Similar to `library/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 `library/rockstock` since you can never run out of sand. If you have plenty of rock and just want the variety, you can import both `library/rockstock` and `library/glassstock` to get a mixture of rock and glass furnishings in your fort.

There are a few items that `library/glassstock` produces that `library/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)

4.6.168 overlay

Tags: dfhack | interface

Command: `overlay`

Manage on-screen overlay widgets.

The overlay framework manages the on-screen widgets that other tools (including 3rd party plugins and scripts) can register for display. For a graphical configuration interface, please see *gui/overlay*. If you are a developer who wants to write an overlay widget, please see the *DFHack overlay dev guide*.

Usage

`enable overlay` Display enabled widgets.

`overlay enable|disable all|<name or list number> [<name or list number> ...]` Enable/disable all or specified widgets. Widgets can be specified by either their name or their number, as returned by `overlay list`.

`overlay list [<filter>]` Show a list of all the widgets that are registered with the overlay framework, optionally filtered by the given filter string.

`overlay position <name or list number> [default|<x> <y>]` Display configuration information for the given widget or change the position where it is rendered. See the *Widget position* section below for details.

`overlay trigger <name or list number>` Intended to be used by keybindings for manually triggering a widget. For example, you could use an `overlay trigger` keybinding to show a menu that normally appears when you hover the mouse over a screen hotspot.

Examples

`overlay enable all` Enable all widgets. Note that they will only be displayed on the screens that they are associated with. You can see which screens a widget will be displayed on, along with whether the widget is a hotspot, by calling `overlay position`.

`overlay position hotkeys.menu` Show the current configuration of the *hotkeys* menu widget.

`overlay position dwarfmonitor.cursor -2 -3` Display the *dwarfmonitor* cursor position reporting widget in the lower right corner of the screen, 2 tiles from the left and 3 tiles from the bottom.

`overlay position dwarfmonitor.cursor default` Reset the *dwarfmonitor* cursor position to its default.

`overlay trigger hotkeys.menu` Trigger the *hotkeys* menu widget so that it shows its popup menu. This is what is run when you hit `CtrlShiftC`.

Widget position

Widgets can be positioned at any (`x`, `y`) position on the screen, and can be specified relative to any edge. Coordinates are 1-based, which means that `1` is the far left column (for `x`) or the top row (for `y`). Negative numbers are measured from the right of the screen to the right edge of the widget or from the bottom of the screen to the bottom of the widget, respectively.

For easy reference, the corners can be found at the following coordinates:

(1, 1) top left corner

(-1, 1) top right corner

(1, -1) lower left corner

(-1, -1) lower right corner

4.6.169 pathable

Tags: dev | inspection | map

Marks tiles that are reachable from the cursor.

This plugin provides a Lua API, but no direct commands. See *pathable* for details.

4.6.170 petcapRemover

Tags: fort | animals

Command: `petcapRemover`

Modify 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 these restrictions and setting your own thresholds. Pets still require PET or PET_EXOTIC tags in order to reproduce. 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

`enable petcapRemover` Enables the plugin and starts running with default settings.

`petcapRemover cap <value>` Set the new population cap per species to the specified value. If set to 0, then there is no cap (good luck with all those animals!). The default cap is 100.

`petcapRemover` Impregnate female pets that have access to a compatible male, up to the population cap.

`petcapRemover every <ticks>` Set how often the plugin will cause pregnancies. The default frequency is every 10,000 ticks (a little over 8 game days).

`petcapRemover pregtime <ticks>` Sets the pregnancy duration to the specified number of ticks. The default value is 200,000 ticks, which is the natural pet pregnancy duration.

4.6.171 plants

Tags: adventure | fort | armok | map | plants

Provides commands that interact with plants.

Command: `plant`

Create a plant or make an existing plant grow up.

Usage

`plant create <ID>` Creates a new plant of the specified type at the active cursor position. The cursor must be on a dirt or grass floor tile.

`plant grow` Grows saplings into trees. If the cursor is active, it only affects the sapling under the cursor. If no cursor is active, it affect all saplings on the map.

To see the full list of plant ids, run the following command:

```
devel/query --table df.global.world.raws.plants.all --search ^id --maxdepth 1
```

Example

`plant create TOWER_CAP` Create a Tower Cap sapling at the cursor position.

4.6.172 plug

Tags: dfhack

Command: `plug`

List available plugins and whether they are enabled.

Usage

```
plug [<plugin> [<plugin> ...]]
```

If run with parameters, it lists only the named plugins. Otherwise it will list all available plugins.

4.6.173 points

Tags: embark | fort | armok

Command: `points`

Sets available points at the embark screen.

Run at the embark screen when you are choosing items to bring with you and skills to assign to your dwarves. You can set the available points to any number.

Usage

```
points <num>
```

Examples

`points 1000000` Grant yourself enough points to buy everything you ever dreamed of.

`points 0` Embark with nothing more than what you have already selected. You can make life quite difficult for yourself this way.

4.6.174 pop-control

Tags: fort | units

Command: `pop-control`

Controls population and migration caps persistently per-fort.

This script controls *hermit* and the various population caps per-fortress. It is intended to be run from `dfhack-config/init/onMapLoad.init` as `pop-control on-load`.

If you edit the population caps using *gui/settings-manager* after running this script, your population caps will be reset and you may get more migrants than you expect.

Usage

`pop-control on-load` Load population settings for this site or prompt the user for settings if not present.

`pop-control reenter-settings` Revise settings for this site.

`pop-control view-settings` Show the current settings for this site.

4.6.175 position

Tags: fort | inspection | map

Command: `position`

Report cursor and mouse position, along with other info.

This tool reports the current date, clock time, month, and season. It also reports the cursor position (or just the z-level if no cursor), window size, and mouse location on the screen.

Usage

```
position
```

4.6.176 power-meter

Tags: fort | gameplay | buildings

Allow pressure plates to measure power.

If you run *gui/power-meter* while building a pressure plate, the pressure plate can be modified to detect power being supplied to gear boxes built in the four adjacent N/S/W/E tiles.

4.6.177 pref-adjust

Tags: fort | armok | units

Command: `pref-adjust`

Set the preferences of a dwarf to an ideal.

This tool replaces a dwarf's preferences with 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.
```

Usage

`pref-adjust all|goth_all|clear_all` Changes/clears preferences for all dwarves.

`pref-adjust one|goth|clear` Changes/clears preferences for the currently selected dwarf.

`pref-adjust list` List all types of preferences. No changes will be made to any dwarves.

Examples

`pref-adjust all` Change preferences for all dwarves to an ideal.

Goth mode

If you select goth mode, this tool will apply the following set of preferences instead of the easy-to-satisfy ideal defaults:

```
... 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.
```

4.6.178 prefchange

Tags: fort | armok | units

Command: `prefchange`

Set strange mood preferences.

This tool sets preferences for strange moods to include a weapon type, equipment type, and material. If you also wish to trigger a mood, see *strangemood*.

Usage

```
prefchange <command>
```

Examples

Examine the preferences across all dwarves:

```
prefchange show
```

Clear a unit's existing preferences and make them like hammers, mail shirts, and steel:

```
prefchange c
prefchange has
```

Commands

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

4.6.179 prioritize

Tags: fort | auto | jobs

Command: `prioritize`

Automatically boost the priority of selected job types.

This tool can force specified types of jobs to get assigned and completed as soon as possible. Finally, you can be sure your food will be hauled before rotting, your hides will be tanned before going bad, and the corpses of your enemies will be cleared from your entranceway expediently.

You can prioritize a bunch of active jobs that you need done *right now*, or you can mark certain job types as high priority, and `prioritize` will watch for and boost the priority of those types of jobs as they are created. This is especially useful for ensuring important (but low-priority – according to DF) jobs don't get ignored indefinitely in busy forts.

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 that `prioritize` is designed to solve. The script provides a good default set of job types to prioritize that have been suggested and playtested by the DF community.

Also see the *do-job-now tweak* and the *do-job-now* script for boosting the priority of specific individual jobs.

Usage

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

Examples

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

`prioritize -a defaults` Prioritize the default set of job types that the community has suggested and playtested (see below for details).

`prioritize -j` Print out the list of active jobs that you can prioritize right now.

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

`prioritize -a --haul-labor=Food,Body StoreItemInStockpile` Prioritize 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.

`-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.

Which job types should I prioritize?

In general, you should prioritize job types that you care about getting done especially quickly and that the game does not prioritize for you. Time-sensitive tasks like food hauling, medical care, and lever pulling are good candidates.

For greater fort efficiency, you should also prioritize jobs that can block the completion of other jobs. For example, dwarves often fill a stockpile up completely, ignoring the barrels, pots, and bins that could be used to organize the items more efficiently. Prioritizing those organizational jobs can mean the difference between having space in your food stockpile for fresh meat and being forced to let it rot in the butcher shop.

It is also convenient to prioritize tasks that block you (the player) from doing other things. When you designate a group of trees for chopping, it's often because you want to *do* something with that space. Prioritizing tree chopping will get your dwarves on the task and keep you from waiting too long.

You may be tempted to automatically prioritize `ConstructBuilding` jobs, but beware that if you engage in megaprojects where many constructions must be built, these jobs can consume your entire fortress if prioritized. It is often better to run `prioritize ConstructBuilding` by itself (i.e. without the `-a` parameter) as needed to just prioritize the construction jobs that you have ready at the time.

Default list of job types to prioritize

The community has assembled a good default list of job types that most players will benefit from. They have been playtested across a wide variety of fort types. Add `prioritize -aq defaults` to your `dfhack-config/init/onMapLoad.init` file to have them automatically prioritized for you in your fort.

They include:

- Handling items that can rot before they rot
- Medical, hygiene, and hospice tasks
- Putting items in bins/barrels/pots/minecarts
- Interactions with animals
- Dumping items, pulling levers, felling trees, and other tasks that you, as a player, might stare at and internally scream "why why why isn't this getting done??".

4.6.180 probe

Tags: adventure | fort | inspection | buildings | map | units

Command: `probe`

Display low-level properties of the selected tile.

Command: `bprobe`

Display low-level properties of the selected building.

Command: `cprobe`

Display low-level properties of the selected unit.

Usage

`probe` Displays properties of the tile selected with `k`. Some of these properties can be passed into *tiletypes*.

`bprobe` Displays properties of the building selected with `q` or `t`. For deeper inspection of the building, see *gui/gm-editor*.

`cprobe` Displays properties of the unit selected with `v`. It also displays the IDs of any worn items. For deeper inspection of the unit and inventory items, see *gui/gm-unit* and *gui/gm-editor*.

4.6.181 prospector

Tags: embark | fort | armok | inspection | map

Provides commands that help you analyze natural resources.

Command: `prospect`

Shows a summary of resources that exist on the map.

It can also calculate an estimate of resources available in the selected embark area.

Usage

```
prospect [all|hell] [<options>]
```

By default, only the visible part of the map is scanned. Include the `all` keyword if you want `prospect` to scan the whole map as if it were revealed. Use `hell` instead of `all` if you also want to see the Z range of HFS tubes in the 'features' report section.

Examples

`prospect all` Shows the entire report for the entire map.

`prospect hell --show layers,ores,veins` Shows only the layers, ores, and other vein stone report sections, and includes information on HFS tubes (if run on a fortress map and not the pre-embark screen).

`prospect all -sores` Show only information about ores for the pre-embark or fortress map report.

Options

`-s`, `--show <sections>` Shows only the named comma-separated list of report sections. Report section names are: summary, liquids, layers, features, ores, gems, veins, shrubs, and trees. If run during pre-embark, only the layers, ores, gems, and veins report sections are available.

`-v`, `--values` Includes material value in the output. Most useful for the 'gems' report section.

Pre-embark estimate

If prospect is called during the embark selection screen, it displays an estimate of layer stone availability. If the `all` keyword is specified, it also estimates ores, gems, and vein material. The estimate covers all tiles of the embark rectangle.

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. In particular, 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.

4.6.182 putontable

Tags: fort | armok | items

Command: `putontable`

Make an item appear on a table.

To use this tool, move an item to the ground on the same tile as a built table. Then, place the cursor over the table and item and run this command. The item will appear on the table, just like in adventure mode shops!

Usage

```
putontable [<options>]
```

Example

`putontable` Of the items currently on the ground under the table, put one item on the table.

`putontable --all` Put all items on the table that are currently on the ground under the table.

Options

`-a`, `--all` Put all items at the cursor on the table, not just one.

4.6.183 questport

Tags: adventure | armok

Command: `questport`

Teleport to your quest log map cursor.

If you open the quest log map and move the cursor to your target location, you can run this command to teleport straight there. This can be done both within and outside of fast travel mode, and it is possible to `questport` in situations where fast travel is normally prohibited.

It is 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.

Usage

```
questport
```

4.6.184 quickfort

Tags: fort | design | productivity | buildings | map | stockpiles

Command: `quickfort`

Apply pre-designed blueprints to your fort.

Quickfort reads blueprint files stored in the `blueprints` subfolder under the main DF game folder and applies the blueprint of your choice to the game map. You can apply blueprints that designate digging, build buildings, place stockpiles, mark zones, and/or configure settings. If you find yourself spending time doing similar or repetitive things in your forts, this tool can be an immense help.

Note that this is the commandline tool. Please see *gui/quickfort* if you'd like a graphical in-game UI for selecting, previewing, and applying blueprints.

You can create the blueprints by hand (see the *Quickfort blueprint creation guide* for details) or you can build your plan "for real" in Dwarf Fortress, and then export your map using *gui/blueprint*. This way you can effectively copy and paste sections of your fort if you need to.

There are many ready-to-use blueprints in the *blueprint library* that is distributed with DFHack, so you can use this tool productively even if you haven't created any blueprints yourself.

Usage

`quickfort list [-m|--mode <mode>] [-u|--useronly] [-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 (please see *Quickfort blueprint creation guide* for more information on modes). By default, blueprints in the `blueprints/library/` subfolder are included and blueprints that contain a `hidden()` marker in their modeline are excluded from the returned list. Specify `-u` or `-h` to exclude library or include hidden blueprints, respectively. The list can additionally 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 reported 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_id>[,<list_id>...] [<options>]` Applies the blueprint(s) with the id number(s) reported 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>` for .xlsx files, 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.

`quickfort set [<key> <value>]` Allows you to modify the global 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 `dfhack-config/quickfort/quickfort.txt`.

<command> is one of:

> **run** Applies the blueprint at your current in-game cursor position.
>
> **orders** Uses the manager interface to queue up workorders to manufacture items needed by 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 and config blueprints since they can contain arbitrary key sequences that are not reversible.

Examples

`quickfort gui library/aquifer_tap.csv -n /dig` Show the in-game preview for the “dig” blueprint in the `library/aquifer_tap.csv` file. You can interactively reposition the blueprint and apply it where you like (it’s intended to be applied in a light aquifer layer – run the associated “help” blueprint for more info).

`quickfort list` List all available blueprints.

`quickfort list dreamfort help` List all the blueprints that have both “dreamfort” and “help” as keywords.

`quickfort run library/dreamfort.csv` Run the first blueprint in the `library/dreamfort.csv` file (which happens to be the blueprint that displays the help).

`quickfort run library/pump_stack.csv -n /dig --repeat up,80 --transform ccw,flipv` Dig a pump stack through 160 z-levels up from the current cursor location (each repetition of the `library/pump_stack.csv -n /dig` blueprint is 2 z-levels). Also transform the blueprint by rotating counterclockwise and flipping vertically in order to fit the pump stack through some tricky-shaped caverns 50 z-levels above. Note that this kind of careful positioning is much easier to do with *gui/quickfort*, but it can be done via the commandline as well if you know exactly what transformations and positioning you need.

`quickfort orders 10,11,12 --dry-run` Process the blueprints with ids `10`, `11`, and `12` (run `quickfort list` to see which blueprints these are for you) and calculate what materials will be needed by your dwarves to actually complete the structures that the blueprints will designate. Display that list to the screen, but don’t actually enqueue the workorders (the `--dry-run` option prevents actual changes to the game).

Command options

`<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 you're trying to figure out why the blueprint isn't being applied like you expect.

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 global 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, and 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 reference*, 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, available from the return value of `reqscript('quickfort)`:

- `quickfort.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 without shifting.

`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 the string `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='d(10x10)',
                          pos=guidm.getCursorPos()}

-- dig a 10x10 block starting at coordinate x=30, y=40, z=50
quickfort.apply_blueprint{mode='dig', data={[50]={[40]={[30]='d(10x10)'}}}}
```

4.6.185 quicksave

Tags: fort

Command: `quicksave`

Immediately save the game.

Keybinding: CtrlAltS in `dwarfmode/Default`

When called in dwarf mode, this tool makes DF immediately save the game by setting a flag normally used for the seasonal auto-save.

Usage

```
quicksave
```

4.6.186 region-pops

Tags: fort | inspection | animals

Command: `region-pops`

Change regional animal populations.

This tool can show or modify the populations of animals in the region.

Usage

`region-pops list [<pattern>]` Shows race populations of the region that your civilization knows about. You can filter the list by specifying a pattern.

`region-pops list-all [<pattern>]` Lists total race populations of the region, including those units that your civilization does not know about. You can filter the list by specifying a pattern.

`region-pops boost <race> <factor>` Multiply all populations of the given race by the given factor. If the factor is greater than one, the command will increase the population. If it is between 0 and 1, the command will decrease the population.

`region-pops boost-all <pattern> <factor>` Same as above, but apply to all races that match the given pattern.

`region-pops incr <race> <amount>` Add the given amount to the population counts of the given race. If the amount is negative, this will decrease the population.

`region-pops incr-all <pattern> <amount>` Same as above, but apply to all races that match the given pattern.

Examples

`region-pops list-all BIRD` List the populations of all the bird races in the region.

`region-pops incr TROLL 1000` Add 1000 trolls to the region.

`region-pops boost-all .* .5` Halve the population of all creatures in the region.

`region-pops boost-all .* 10` Increase populations of all creatures by a factor of 10. Hashtag zoolife.

4.6.187 regrass

Tags: adventure | fort | armok | animals | map

Command: `regrass`

Regrow all the grass.

Use this command if your grazers have eaten everything down to the dirt.

Usage

```
regrass [max]
```

Specify the 'max' keyword to pack more grass onto a tile than what the game normally allows to give your grazers extra chewing time.

4.6.188 rejuvenate

Tags: fort | armok | units

Command: `rejuvenate`

Sets unit age to 20 years.

If your most valuable citizens are getting old, this tool can save them. It decreases the age of the selected dwarf to 20 years.

Usage

```
rejuvenate [<options>]
```

Examples

`rejuvenate` Set the age of the selected dwarf to 20 (if they're older).

`rejuvenate --all` Set the age of all dwarves over 20 to 20.

`rejuvenate --all --force` Set the age of all dwarves (including babies) to 20.

Options

--all Rejuvenate all citizens, not just the selected one.

--force Set age for units under 20 years old to 20 years.. Useful if there are too many babies around. . .

--dry-run Only list units that would be changed; don't actually change ages.

4.6.189 reload

Tags: dfhack

Command: `reload`

Reload a loaded plugin.

Developers use this command to reload a plugin that they are actively modifying. Also see *load* and *unload* for related actions.

Usage

```
reload <plugin> [<plugin> ...]
reload -a|--all
```

You can reload individual named plugins or all plugins at once. Note that plugins are disabled after loading/reloading until you explicitly *enable* them.

4.6.190 remove-stress

Tags: fort | armok | units

Command: `remove-stress`

Reduce stress values for fortress dwarves.

Generally happy dwarves have stress values in the range of 0 to 500,000. If they encounter things that stress them out, or if their needs are not being met, that value will increase. When it increases too high, your dwarves will start to have negative repercussions. This tool can magically whisk away some (or all) of their stress so they can function normally again.

Usage

```
remove-stress [--all] [--value <value>]
```

Examples

`remove-stress` Makes the currently selected dwarf blissfully unstressed.

`remove-stress --all` Makes all dwarves blissfully unstressed.

`remove-stress --all --value 10000` Reduces stress to 10,000 for all dwarves whose stress value is currently above that number.

Options

`--all` Apply to all dwarves instead of just the currently selected dwarf.

`--value <value>` Decrease stress level to the given value. If the value is negative, prepend the negative sign with a backslash (e.g. `--value \-50,000`).

4.6.191 remove-wear

Tags: fort | armok | items

Command: `remove-wear`

Remove wear from items in your fort.

If your clothes are all wearing out and you wish you could just repair them instead of having to make new clothes, then this tool is for you! This tool will set the wear on items in your fort to zero, as if they were new.

Usage

`remove-wear all` Remove wear from all items in your fort.

`remove-wear <item id> ...` Remove wear from items with the given ID numbers.

You can discover the ID of an item by selecting it in the UI and running the following command:

```
:lua !item.id
```

4.6.192 rename

Tags: adventure | fort | productivity | buildings | stockpiles | units

Command: `rename`

Easily rename things.

Use *gui/rename* for an in-game interface.

Usage

`rename squad <ordinal> "<name>"` Rename the indicated squad. The ordinal is the number that corresponds to the list of squads in the squads menu (`s`). The first squad is ordinal `1`.

`rename hotkey <ordinal> "<name>"` Rename the indicated hotkey. The ordinal the the number that corresponds to the list of hotkeys in the hotkeys menu (`H`). The first hotkey is ordinal `1`.

`rename unit "<name>"` Give the selected unit the given nickname.

`rename unit-profession "<name>"` Give the selected unit the given profession name.

`rename building "<name>"` Set a custom name to the selected building. The building must be a stockpile, workshop, furnace, trap, siege engine, or activity zone.

Example

`rename squad 1 "The Whiz Bonkers"` Rename the first squad to The Whiz Bonkers.

4.6.193 rendermax

Tags: adventure | fort | gameplay | graphics

Command: `rendermax`

Modify the map lighting.

This plugin provides a collection of OpenGL lighting filters that affect how the map is drawn to the screen.

Usage

`rendermax light` Light the map tiles realistically. Outside tiles are light during the day and dark at night. Inside tiles are always dark unless a nearby unit is lighting it up, as if they were carrying torches.

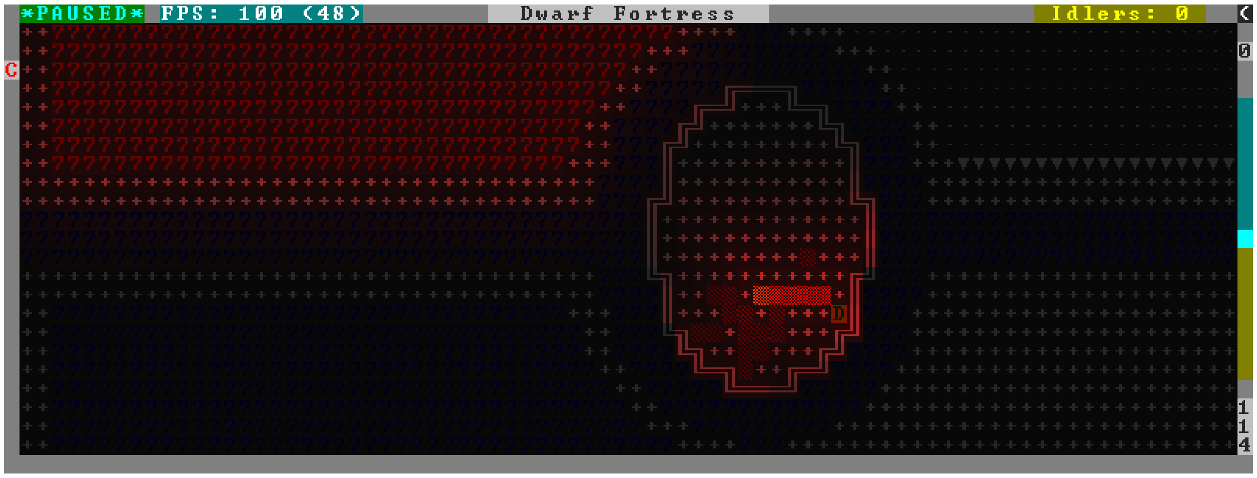
`rendermax light sun <hour>|cycle` Set the outside lighting to correspond with the specified day hour (1-24), or specify `cycle` to have the lighting follow the sun (which is the default).

`rendermax light reload` Reload the lighting settings file.

`rendermax trippy` Randomize the color of each tile. Used for fun, or testing.

`rendermax disable` Disable any `rendermax` lighting filters that are currently active.

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



For better visibility, try changing the black color in palette to non totally black. See Bay12 forums thread 128487 for more info.

4.6.194 repeat

Tags: dfhack

Command: `repeat`

Call a DFHack command at a periodic interval.

You can use this utility command to periodically call other DFHack commands. This is especially useful for setting up “maintenance” commands that you want called every once in a while but don’t want to have to remember to run yourself.

Usage

`repeat [--name <name>] --time <delay> [--timeUnits <units>] --command [ <command> ]` Register the given command to be run periodically.

`repeat --list` Show the currently registered commands and their names.

`repeat --cancel <name>` Unregister the given registered command.

Examples

`repeat --name orders-sort --time 600 --command [ orders sort ]` Sort your manager workorders every 600 ticks (about half a day).

`repeat --time 10 --timeUnits days --command [ warn-starving ]` Check for starving dwarves and pets every 10 game days.

`repeat --cancel warn-starving` Unregister the warn-starving command registered above.

Options

`--name <name>` Registers the command under the given name. This ensures you have a memorable identifier for the `--list` output so you can unregister the command if you want. It also prevents the same command from being registered twice. If not specified, it's set to the first argument after `--command`.

`--time <delay>` Sets the delay between invocations of the command. It must be a positive integer.

`--timeUnits <units>` A unit of time for the value passed with the `--time` option. Units can be `frames` (raw FPS), `ticks` (unpaused game frames), or the in-world time measurements of `days`, `months`, or `years`. If not specified, `ticks` is the default.

`--command [ ... ]` The `...` specifies the command to be run, just as you would write it on the commandline. Note that the command must be enclosed in square brackets.

Registering commands

It is common that you want to register the same set of commands every time you load a game. For this, it is convenient to add the `repeat` commands you want to run to the `dfhack-config/init/onMapLoad.init` file so they are run whenever you load a fort.

4.6.195 resurrect-adv

Tags: adventure | armok

Command: `resurrect-adv`

Bring a dead adventurer back to life.

Have you ever died, but wish you hadn't? This tool can help :) When you see the "You are deceased" message, run this command to be resurrected and fully healed.

Usage

```
resurrect-adv
```

Note that you can only resurrect the current player character in your party, and you must run this tool immediately after dying. It is not possible to resurrect the adventurer after the game has been ended.

4.6.196 reveal

Tags: adventure | fort | armok | inspection | map

Command: `reveal`

Reveals the map.

Command: `unreveal`

Hides previously hidden tiles again.

Command: `revforget`

Discard records about what was visible before revealing the map.

Command: `revtoggle`

Switch between reveal and unreveal.

Command: `revflood`

Hide everything, then reveal tiles with a path to the cursor.

Command: `nopause`

Disable pausing.

This reveals all z-layers in fort mode. It 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` to hide the map again.

Usage

`reveal [hell|demon]` Reveal the whole map. If `hell` is specified, also reveal HFS areas, but you are required to run `unreveal` before unpausing is allowed in order to prevent the demons from spawning. If you really want to unpause with hell revealed, specify `demon` instead of `hell`.

`unreveal` Reverts the effects of `reveal`.

`revtoggle` Switches between `reveal` and `unreveal`. Convenient to bind to a hotkey.

`revforget` Discard info about what was visible before revealing the map. Only useful where (for example) you abandoned with the fort revealed and no longer need the saved map data when you load a new fort.

`revflood` Hide everything, then reveal tiles with a path to the cursor. This allows reparing maps that you accidentally saved while they were revealed. Note that tiles behind constructed walls are also revealed as a workaround for Bug 1871.

`nopause 1|0` Disables pausing (both manual and automatic) with the exception of the pause forced by *reveal* `hell`. This is nice for digging under rivers. Use `nopause 1` to prevent pausing and `nopause 0` to allow pausing like normal.

4.6.197 reveal-adv-map

Tags: adventure | armok | map

Command: `reveal-adv-map`

Reveal or hide the world map.

This tool 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 traveling).

Note that hidden lairs, camps, etc. are not revealed. Please see *reveal-hidden-sites* for that functionality.

Usage

`reveal-adv-map` Reveal all world tiles.

`reveal-adv-map --hide` Hide all world tiles.

4.6.198 reveal-hidden-sites

Tags: adventure | armok | map

Command: `reveal-hidden-sites`

Reveal all sites in the world.

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

It is usable in both fortress and adventure mode.

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

Usage

```
reveal-hidden-sites
```

4.6.199 reveal-hidden-units

Tags: adventure | fort | armok | map | units

Command: `reveal-hidden-units`

Reveal sneaking units.

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

It is usable in both fortress and adventure mode.

Usage

```
reveal-hidden-units
```

4.6.200 ruby

Tags: dev

Allow Ruby scripts to be executed as DFHack commands.

Command: `rb`

Eval() a ruby string.

Command: `rb_eval`

Eval() a ruby string.

Usage

```
enable ruby
rb "ruby expression"
rb_eval "ruby expression"
:rb ruby expression
```

Example

:rb puts df.unit_find(:selected).name Print the name of the selected unit.

4.6.201 sc-script

Tags: dfhack

Command: `sc-script`

Run commands when game state changes occur.

This is similar to the static *Init files* but is slightly more flexible since it can be set dynamically.

Usage

sc-script [help] Show the list of valid event names.

sc-script list [<event>] List the currently registered files for all events or the specified event.

sc-script add|remove <event> <file> [<file> ...] Register or unregister a file to be run for the specified event.

Example

sc-script add SC_MAP_LOADED spawn_extra_monsters.init Registers the `spawn_extra_monsters.init` file to be run whenever a new map is loaded.

4.6.202 script

Tags: dfhack

Command: `script`

Execute a batch file of DFHack commands.

It 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 can later be reloaded with `script`.

Usage

```
script <filename>
```

Example

`script startup.txt` Executes the commands in `startup.txt`, which exists in your DF game directory.

4.6.203 search

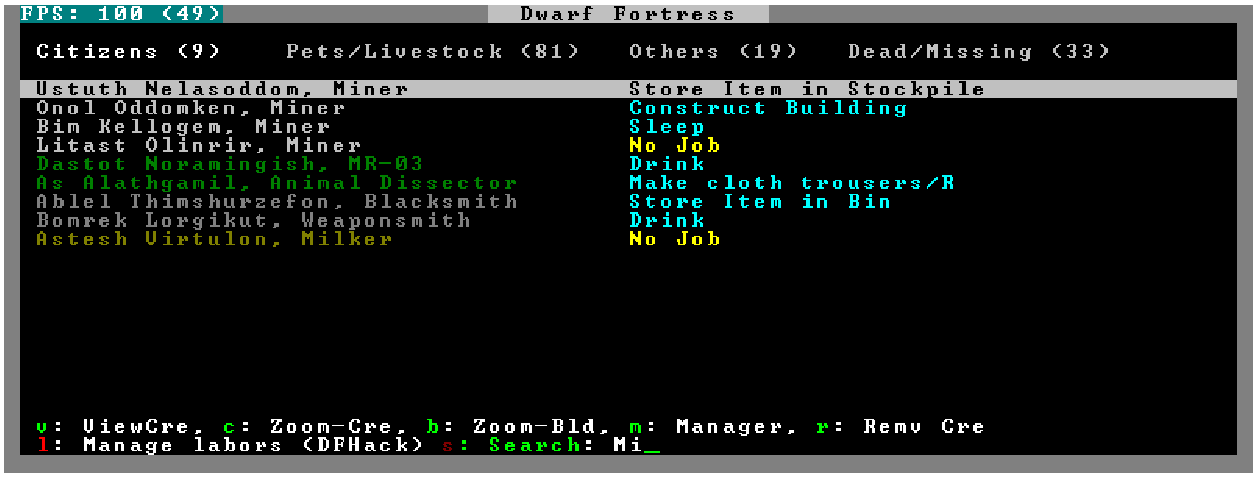
Tags: fort | productivity | interface

Adds search capabilities to the UI.

Search options are added to the Stocks, Animals, Trading, Stockpile, Noble assignment candidates), Military (position candidates), Burrows (unit list), Rooms, Announcements, Job List, and Unit List screens all get hotkeys that allow you to dynamically filter the displayed lists.

Usage

```
enable search
```



```
FPS: 100 (49)                 Dwarf Fortress

Citizens (9)    Pets/Livestock (81)    Others (19)    Dead/Missing (33)

Ustuth Nelasoddom, Miner                Store Item in Stockpile
Onol Oddomken, Miner                    Construct Building
Bim Kellogem, Miner                     Sleep
Litast Olinrir, Miner                   No Job
Dastot Noramingish, MR-03               Drink
As Alathgamil, Animal Dissector         Make cloth trousers/R
Ablel Thimshurzefon, Blacksmith         Store Item in Bin
Bomrek Lorgikut, Weaponsmith            Drink
Astesh Virtulon, Milker                 No Job

v: ViewCre, c: Zoom-Cre, b: Zoom-Bld, m: Manager, r: Remv Cre
l: Manage labors (DFHack) s: Search: Mi_
```

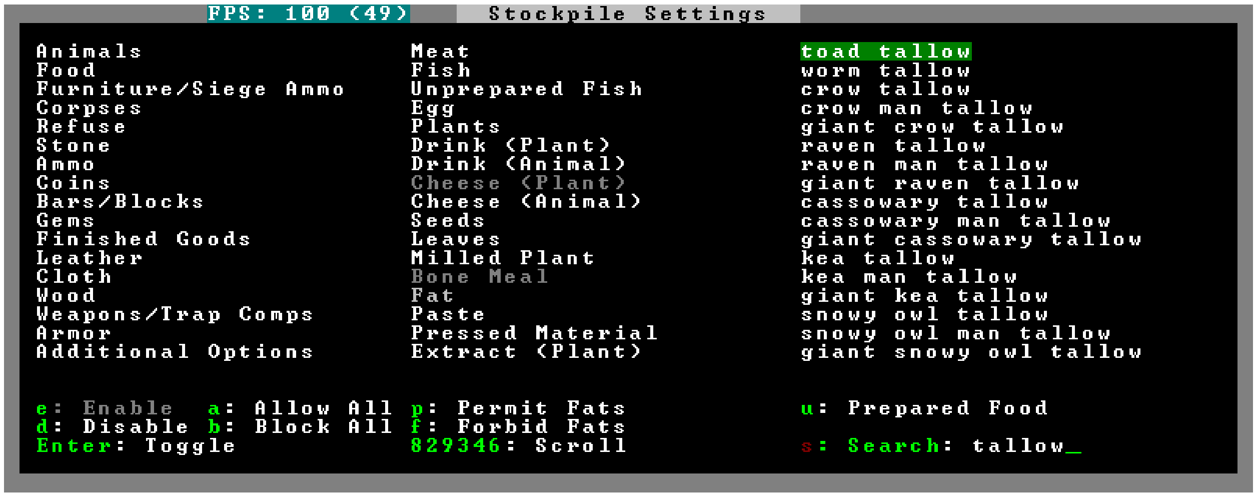
Searching works the same way as the search option in *Move to Depot*. You will see the Search option displayed on screen with a hotkey (usually `s`). Pressing it lets you start typing a query and the relevant list will start filtering automatically.

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:



```
        FPS: 100 (49)      Stockpile Settings

 Animals                  Meat                 toad tallow
 Food                     Fish                 worm tallow
 Furniture/Siege Ammo     Unprepared Fish      crow tallow
 Corpses                  Egg                  crow man tallow
 Refuse                   Plants               giant crow tallow
 Stone                    Drink (Plant)        raven tallow
 Ammo                     Drink (Animal)       raven man tallow
 Coins                    Cheese (Plant)       giant raven tallow
 Bars/Blocks              Cheese (Animal)      cassowary tallow
 Gems                     Seeds                cassowary man tallow
 Finished Goods           Leaves               giant cassowary tallow
 Leather                  Milled Plant         kea tallow
 Cloth                    Bone Meal            kea man tallow
 Wood                     Fat                  giant kea tallow
 Weapons/Trap Comps       Paste                snowy owl tallow
 Armor                    Pressed Material     snowy owl man tallow
 Additional Options       Extract (Plant)      giant snowy owl tallow

 e: Enable   a: Allow All p: Permit Fats       u: Prepared Food
 d: Disable  b: Block All f: Forbid Fats
 Enter: Toggle            829346: Scroll       s: Search: tallow_
```

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.

4.6.204 season-palette

Tags: fort | gameplay | graphics

Command: `season-palette`

Swap color palettes when the seasons change.

For this tool to work you need to add *at least* one color palette file to your save raw directory. These files must be in the same format as `data/init/colors.txt`.

Palette file names are:

```
"colors.txt": The world (worldgen and default 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 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 is used 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

`enable season-palette` Begin swapping seasonal color palettes.

`disable season-palette` Stop swapping seasonal color palettes and load the default color palette.

API

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. Otherwise, it returns true.

4.6.205 seedwatch

Tags: fort | auto | plants

Command: `seedwatch`

Manages seed and plant cooking based on seed stock levels.

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

The plugin needs a fortress to be loaded and will deactivate automatically otherwise. You have to reactivate with `enable seedwatch` after you load a fort.

Usage

`enable seedwatch` Start managing seed and plant cooking. By default, no types are watched. You have to add them with further `seedwatch` commands.

`seedwatch <type> <target>` Adds the specified type to the watchlist (if it's not already there) and sets the target number of seeds to the specified number. You can pass the keyword `all` instead of a specific type to set the target for all types.

`seedwatch <type>` Removes the specified type from the watch list.

`seedwatch clear` Clears all types from the watch list.

`seedwatch info` Display whether seedwatch is enabled and prints out the watch list.

To print out a list of all plant types, you can run this command:

```
devel/query --table df.global.world.raws.plants.all --search ^id --maxdepth 1
```

Examples

`seedwatch all 30` Adds all seeds to the watch list and sets the targets to 30.

`seedwatch MUSHROOM_HELMET_PLUMP 50` Add Plump Helmets to the watch list and sets the target to 50.

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

4.6.206 set-orientation

Tags: fort | armok | units

Command: `set-orientation`

Alter a unit's romantic inclinations.

This tool lets you tinker with the interest levels your dwarves have towards dwarves of the same/different sex.

Usage

`set-orientation [--unit <id>] --view` See the unit's current orientation values.

`set-orientation [--unit <id>] <interest options>` Set the orientation values for the unit.

If a unit id is not specified or is not found, the default is to target the currently selected unit.

Examples

`set-orientation --male 0 --female 0` Make a dwarf romantically inaccessible

`set-orientation --random` Re-randomize the orientation values for this dwarf.

Interest options

Interest levels are 0 for Uninterested, 1 for Romance, and 2 for Marry.

`--male <INTEREST>` Set the interest level towards males.

`--female <INTEREST>` Set the interest level towards females.

`--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.

4.6.207 set-timeskip-duration

Tags: adventure | embark | fort | armok

Command: `set-timeskip-duration`

Modify the duration of the pre-game world update.

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 tool 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)
- the timeskip duration is overwritten by setting a new duration

Usage

```
set-timeskip-duration --clear
set-timeskip-duration <duration options>
```

Examples

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

Options

The <num> values passed to any option below must be positive integers (or 0).

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

`--ticks <num>` 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
```

(continues on next page)

(continued from previous page)

```
33600 ticks = 4 weeks = 1 month
403200 ticks = 12 months = 1 year
```

--years <num>, --months <num>, --days <num>, --hours <num> Adds the appropriate number ticks to the timeskip duration

4.6.208 setfps

Tags: fps

Command: `setfps`

Set the graphics FPS cap.

This command can set the FPS cap at runtime. This is useful for when you want to speed up the game or watch combat in slow motion.

Note that setting the cap higher than what your computer can support will not make the game go any faster.

Usage

```
setfps <number>
```

4.6.209 show

Tags: dfhack

Command: `show`

Unhides the DFHack terminal window.

Useful if you have hidden the terminal with *hide* and you want it back. Since the terminal window won't be available to run this command, you'll need to use it from a *keybinding* set beforehand or the in-game *command-prompt*.

Only available on Windows.

Usage

```
show
```

4.6.210 show-unit-syndromes

Tags: fort | inspection | units

Command: `show-unit-syndromes`

Inspect syndrome details.

This tool can list the syndromes affecting game units and the remaining and maximum duration of those syndromes, along with (optionally) substantial detail about the effects.

Usage

`show-unit-syndromes selected|dwarves|livestock|wildanimals|hostile [<options>]` Shows information for the specified category of units.

`show-unit-syndromes world [<options>]` Shows information about all possible syndromes in the world.

Examples

`show-unit-syndromes selected` Show a summary of the syndromes affecting the selected unit.

`show-unit-syndromes dwarves showall` Show a summary of the syndromes affecting all citizens, including the citizens who are not afflicted with any syndromes.

`show-unit-syndromes world showeffects export:allsyndromes.txt` Export a detailed description of all the world's syndromes into the `allsyndromes.txt` file.

Options

`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.

`export:<filename>` Send output to the given file instead of the console.

4.6.211 showmood

Tags: fort | armok | inspection | jobs | units

Command: `showmood`

Shows all items needed for the active strange mood.

Usage

```
showmood
```

4.6.212 siege-engine

Tags: fort | gameplay | buildings

Extend the functionality and usability of siege engines.

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

Usage

```
enable siege-engine
```

You can use the new features by selecting a built siege engine and running *gui/siege-engine*.

4.6.213 siren

Tags: fort | armok | units

Command: `siren`

Wake up sleeping units and stop parties.

Sound the alarm! This tool can shake your sleeping units awake and knock some sense into your party animal military dwarves so they can address a siege.

Note that this is not without consequences. Affected dwarves will receive a negative thought about noise, tiredness, and lack of protection.

Usage

```
siren [<burrow> ...]
```

Examples

`siren` Wake up the whole fort!

`siren barracks tavern` Just affect dwarves that are within the bounds of the `barracks` and `tavern` burrows (note that the dwarves do not need to be *assigned* to those burrows).

4.6.214 sort

Tags: fort | productivity | interface

Sort lists shown in the DF interface.

Command: `sort-items`

Sort the visible item list.

Command: `sort-units`

Sort the visible unit list.

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"`

Usage

```
sort-items <property> [<property> ...]
sort-units <property> [<property> ...]
```

Both commands sort the visible list using the given sequence of comparisons. Each property can be prefixed with a < or > character to indicate whether elements that don't have the given property defined go first or last (respectively) in the sorted list.

Examples

`sort-items material type quality` Sort a list of items by material, then by type, then by quality

`sort-units profession name` Sort a list of units by profession, then by name

Properties

Items can be sorted by the following properties:

- `type`
- `description`
- `base_quality`
- `quality`
- `improvement`
- `wear`
- `material`

Units can be sorted by the following properties:

- `name`
- `age`
- `arrival`
- `noble`
- `profession`
- `profession_class`
- `race`
- `squad`
- `squad_position`
- `happiness`

4.6.215 source

Tags: fort | armok | map

Command: `source`

Create an infinite magma or water source.

This tool can create an infinite magma or water source or drain on a tile. For more complex liquid placement, try *liquids* or *gui/liquids*.

Map tiles registered with this tool as a liquid source will be set to have the configured amount of liquid every 12 game ticks. A standard liquid source sets the level to `7`, and a standard drain sets the level to `0`, but you can set the target anywhere in between as well.

Usage

`source add water|magma [0-7]` Add a source or drain at the selected tile position. If the target level is not specified, it defaults to `7`. The cursor must be over a flow-passable tile (e.g. empty space, floor, staircase, etc.) and not too high in the sky.

`source list` List all currently registered source tiles.

`source delete` Remove the source under the cursor.

`source clear` Remove all liquid sources that have been added with `source`.

Examples

`source add water` Create an infinite water source under the cursor.

`source add water 3` Create an infinite water source under the cursor, but equalize the water depth at 3/7. This is useful when creating a swimming pool for your dwarves.

`source add water 0` Create a water drain under the cursor.

`source add magma` Create an infinite magma source under the cursor.

4.6.216 spawnunit

Tags: fort | armok | units

Command: `spawnunit`

Create a unit.

This tool allows you to easily spawn a unit of your choice. It is a simplified interface to *modtools/create-unit*, which this tool uses to actually create the requested unit.

Usage

```
spawnunit [-command] <race> <caste> [<name> [<x> <y> <z>]] [...]
```

If `-command` is specified, the generated *modtools/create-unit* command is printed to the terminal instead of being run.

The name and coordinates of the unit are optional. Any further arguments are simply passed on to *modtools/create-unit*. See documentation for that tool for information on what you can pass through.

To see the full list of races and castes for your world, run the following command:

```
dfhack-run devel/query --table df.global.world.raws.creatures.all --search [ creature_id␣
↪caste_id ] --maxdepth 3 --maxlength 5000
```

Examples

`spawnunit GOBLIN MALE` Warp in a (male) goblin for your squads to beat on.

`spawnunit JABBERER FEMALE --domesticate` Spawn a tame female jabberer for breeding an army!

4.6.217 spectate

Tags: fort | interface

Command: `spectate`

Automatically follow productive dwarves.

Usage

```
enable spectate
spectate set <setting> <value>
spectate enable|disable <feature>
```

When enabled, the plugin will automatically switch which dwarf is being followed periodically, preferring dwarves on z-levels with the highest job activity.

Changes to plugin settings will be saved per world. Whether the plugin itself is enabled or not is not saved.

Examples

`spectate` The plugin reports its configured status.

`spectate enable auto-unpause` Enable the spectate plugin to automatically dismiss pause events caused by the game. Siege events are one example of such a game event.

`spectate set tick-threshold 50` Set the tick interval the followed dwarf can be changed at back to its default value.

Features

auto-unpause Toggle auto-dismissal of game pause events. (default: disabled)

auto-disengage Toggle auto-disengagement of plugin through player intervention while unpaused. (default: disabled)

animals Toggle whether to sometimes follow animals. (default: disabled)

hostiles Toggle whether to sometimes follow hostiles (eg. undead, titan, invader, etc.) (default: disabled)

visiting Toggle whether to sometimes follow visiting units (eg. diplomat)

Settings

tick-threshold Set the plugin's tick interval for changing the followed dwarf. (default: 1000)

4.6.218 startdwarf

Tags: embark | fort | armok

Command: `startdwarf`

Increase the number of dwarves you embark with.

You must use this tool before embarking (e.g. at the site selection screen or any time before) to change the number of dwarves you embark with from the default of 7.

Note that the game requires that you embark with no fewer than 7 dwarves, so this tool can only increase the starting dwarf count, not decrease it.

Usage

```
startdwarf <number>
```

Examples

`startdwarf 10` Start with a few more warm bodies to help you get started.

`startdwarf 500` Start with a teeming army of dwarves (leading to immediate food shortage and FPS issues).

4.6.219 starvingdead

Tags: fort | auto | fps | gameplay | units

Command: `starvingdead`

Prevent infinite accumulation of roaming undead.

With this tool 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.

In any game, this can be a welcome gameplay feature, but it is especially useful in preventing undead cascades in the caverns, where constant combat can lead to hundreds of undead roaming the caverns and destroying your FPS.

Usage

```
starvingdead start
starvingdead stop
```

4.6.220 steam-engine

Tags: fort | gameplay | buildings

Allow modded steam engine buildings to function.

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

The plugin auto-enables itself when it detects the relevant tags in the world raws. It does not need to be enabled with the *enable* command.

Rationale

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 dwarven 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 DF game 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 DF game 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, or 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% is 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.

A masterpiece mechanism and chain will decrease the mechanical power drawn by the engine itself from 10 to 5. A masterpiece barrel decreases waste rate by 4%. A 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 gradually wear 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, of course, 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.

4.6.221 stockflow

Tags: fort | auto | stockpiles | workorders

Command: `stockflow`

Queue manager jobs based on free space in stockpiles.

With this plugin, the fortress bookkeeper can tally up free space in specific stockpiles and queue jobs through the manager to produce items to fill the free space.

When the plugin is 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.

This plugin is similar to *workflow*, but uses stockpiles to manage job triggers instead of abstract stock quantities.

Usage

`enable stockflow` Enable the plugin.

`stockflow status` Display whether the plugin is enabled.

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

`stockflow fast` Enqueue orders once per day instead of waiting for the bookkeeper.

4.6.222 stockpiles

Tags: fort | design | productivity | stockpiles

Import and export stockpile settings.

Command: `copystock`

Copies the configuration of the selected stockpile.

Keybinding: AltP in `dwarfmode/QueryBuilding/Some/Stockpile`

Command: `savestock`

Exports the configuration of the selected stockpile.

Command: `loadstock`

Imports the configuration of the selected stockpile.

When the plugin is enabled, the `q` menu of each stockpile will have an option for saving or loading the stockpile settings. See *gui/stockpiles* for an in-game interface.

Usage

`enable stockpiles` Add a hotkey that you can hit to easily save and load settings from stockpiles selected in `q` mode.

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

`savestock <filename>` 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.

`loadstock <filename>` Loads a saved stockpile settings file and applies it to the currently selected stockpile.

Filenames with spaces are not supported. Generated materials, divine metals, etc. are not saved as they are different in every world.

Examples

`savestock food_settings.dfstock` Export the stockpile settings for the stockpile currently selected in `q` mode to a file named `food_settings.dfstock`.

`loadstock food_settings.dfstock` Set the selected stockpile settings to those saved in the `food_settings.dfstock` file.

4.6.223 stocks

Tags: fort | productivity | items

Command: `stocks`

Enhanced fortress stock management interface.

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

When the plugin is enabled, two new hotkeys become available:

- `e` on the vanilla DF stocks screen (`z` and then select Stocks) will launch the fortress-wide stock management screen.
- `i` when a stockpile is selected in `q` mode will launch the stockpile inventory management screen.

Usage

```
enable stocks
stocks show
```

Running `stocks show` will bring you to the fortress-wide stock management screen from wherever you are.

4.6.224 stonesense

Tags: adventure | fort | graphics | map

Command: `stonesense`

A 3D isometric visualizer.

Command: `ssense`

An alias for stonesense.

Usage

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

`ssense overlay` Overlay DF window, replacing the map area.

The viewer window has read-only access to the game, and can follow the game view or be moved independently. Configuration for stonesense can be set in the `stonesense/init.txt` file in your DF game directory. If the window refresh rate is too low, change `SEGMENTSIZE_Z` to `2` in this file, and if you are unable to see the edges of the map with the overlay active, try decreasing the value for `SEGMENTSIZE_XY` – normal values are `50` to `80`, depending on your screen resolution.

If you replace the map section of your DF window with `ssense overlay`, be aware that it's not (yet) suitable for use as your only interface. Use DF's `[PRINT_MODE:2D]` init option (in `data/init/init.txt`) for stability.

Controls

Mouse controls are hard-coded and cannot be changed.

Left click Move debug cursor (if available)

Right click Recenter screen

Scrollwheel Move up and down

Ctrl-Scroll Increase/decrease Z depth shown

Follow mode makes the Stonesense view follow the location of the DF window. The offset can be adjusted by holding `Ctrl` while using the keyboard window movement keys. When you turn on cursor follow mode, the Stonesense debug cursor will follow the DF cursor when the latter exists.

You can take screenshots with `F5`, larger screenshots with `CtrlF5`, and screenshot the whole map at full resolution with `CtrlShiftF5`. Screenshots are saved to the DF directory. Note that feedback is printed to the DFHack console, and you may need to zoom out before taking very large screenshots.

See `stonesense/keybinds.txt` to learn or set keybindings, including zooming, changing the dimensions of the rendered area, toggling various views, fog, and rotation. Here's the important section:

```
INSTRUCTIONS:

This document specifies the keys and associated actions stonesense
can recognize.  The syntax is:
        [<action name>:<action key 1>:<action key 2> ... ]
If the closing brace is preceeded by an asterisk:
        [<stuff>*]
then the keys specified will repeat each frame until released,
otherwise it will occur exactly once each time the character is
registered.

It is possible to specify the same action on multiple lines, or
to leave action names out of the list completely.

Listing multiple actions on the same line is not supported.
Likewise listing the same key for multiple actions will result
in only the last action listed being the one taken when the key is
pressed.  Currently only keyboard events are supported;
stonesense's mouse events are all hardcoded.

A complete listing of valid actions and key values can be found at
```

Fig. 1: The above-ground part of the fortress *Roadtruss*.

```
the bottom of this file.

KEYBINDINGS:

[ROTATE:KEYS_ENTER]
[RELOAD_SEGMENT:KEY_R]
[TOGGLE_DESIGNATIONS:KEY_D]
[TOGGLE_STOCKS:KEY_I]
[TOGGLE_ZONES:KEY_U]
[TOGGLE_OCCLUSION:KEY_O]
[TOGGLE_CREATURE_MOODS:KEY_M]
[TOGGLE_CREATURE_PROFS:KEY_P]
[TOGGLE_CREATURE_JOBS:KEY_J]
[TOGGLE_CREATURE_NAMES:KEY_N]
[CHOP_WALLS:KEY_C]
[CYCLE_TRACKING_MODE:KEY_F]
[RESET_VIEW_OFFSET:KEY_Z]
[DECR_SEGMENT_Z:KEY_1]
[INCR_SEGMENT_Z:KEY_2]
[TOGGLE_SINGLE_LAYER:KEY_S]
[TOGGLE_SHADE_HIDDEN_TILES:KEY_B]
[TOGGLE_SHOW_HIDDEN_TILES:KEY_H]
[TOGGLE_OSD:KEYF_2]
[TOGGLE_KEYBINDS:KEYS_SLASH]
[INCR_ZOOM:KEYS_FULLSTOP]
[DECR_ZOOM:KEYS_COMMA]
[SCREENSHOT:KEYF_5]
[INCR_RELOAD_TIME:KEYPAD_PLUS]
[DECR_RELOAD_TIME:KEYPAD_MINUS]
[CREDITS:KEYF_9]

[DECR_Y:KEYS_UP*]
[INCR_Y:KEYS_DOWN*]
[DECR_X:KEYS_LEFT*]
[INCR_X:KEYS_RIGHT*]
[DECR_Z:KEYS_PGDN:KEY_9]
[INCR_Z:KEYS_PGUP:KEY_0]
```

Known Issues

If Stonesense gives an error saying that it can't load `creatures/large_256/*.png`, your video card cannot handle the high detail sprites used. Either open `creatures/init.txt` and remove the line containing that folder, or use these smaller sprites.

Stonesense requires working graphics acceleration, and we recommend at least a dual core CPU to avoid slowing down your game of DF.

Useful links

- Official Stonesense thread for feedback, questions, requests or bug reports
- Screenshots thread
- Main wiki page
- How to add content
- Stonesense on Github

4.6.225 strangemood

Tags: fort | armok | units

Command: `strangemood`

Trigger a strange mood.

Usage

```
stangemood [<options>]
```

Examples

`strangemood -force -unit -type secretive -skill armorsmith` Trigger a strange mood for the selected unit that will cause them to become a legendary armorsmith.

Options

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

`-unit` Make the strange mood strike the selected unit instead of picking one randomly. Unit eligibility is still enforced (unless `-force` is also specified).

`-type <type>` Force the mood to be of a particular type instead of choosing randomly based on happiness. Valid values are “fey”, “secretive”, “possessed”, “fell”, and “macabre”.

`-skill <skill>` 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”, “leather-worker”, “bonecarver”, “bowyer”, and “mechanic”.

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

4.6.226 stripcaged

Tags: fort | productivity | items

Command: `stripcaged`

Remove items from caged prisoners.

This tool helps with the tedious task of going through all your cages and marking the items inside for dumping. This lets you get leftover seeds out of cages after you tamed the animals inside. The most popular use of this tool, though, is to strip the weapons and armor from caged prisoners. After you run this tool, your dwarves will come and take the items to the garbage dump, leaving your cages clean and your prisoners stripped bare.

If you don't want to wait for your dwarves to dump all the items, you can use *autodump* to speed the process along.

Usage

`stripcaged list` Display a list of all cages and their item contents.

`stripcaged items|weapons|armor|all [here|<cage id> ...]` Dump the given type of item. If `here` is specified, only act on the in-game selected cage (or the cage under the game cursor). Alternately, you can specify the item ids of specific cages that you want to target.

Examples

`stripcaged all` For all cages, dump all items, equipped by a creature or not.

`stripcaged items` Dump loose items in all cages, such as seeds left over from animal training.

`stripcaged weapons` Dump weapons equipped by caged creatures.

`stripcaged armor here` Dumps the armor equipped by the caged creature under the cursor.

`stripcaged all 25321 34228` Dumps all items out of the specified cages.

4.6.227 superdwarf

Tags: fort | armok | units

Command: `superdwarf`

Make a dwarf supernaturally speedy.

Select a dwarf in-game and run this tool to make them super fast. They will complete tasks instantly and never need to rest.

Usage

`superdwarf add` Give superspeed to selected creature.

`superdwarf del` Remove superspeed from selected creature.

`superdwarf clear` Remove superspeed from all creatures.

`superdwarf list` List creatures with superspeed.

4.6.228 tags

Tags: dfhack

Command: `tags`

List the categories of DFHack tools or the tools with those tags.

DFHack tools are labeled with tags so you can find groups of related commands. This builtin command lists the tags that you can explore, or, if called with the name of a tag, lists the tools that have that tag.

Usage

`tags` List the categories of DFHack tools and a description of those categories.

`tags <tag>` List the tools that are tagged with the given tag.

Examples

`tags` List the defined tags.

`tags design` List all the tools that have the `design` tag.

4.6.229 tailor

Tags: fort | auto | workorders

Command: `tailor`

Automatically keep your dwarves in fresh clothing.

Whenever the bookkeeper updates stockpile records, this plugin will scan the fort. If there are fresh cloths available, dwarves who are wearing tattered clothing will have their rags confiscated (in the same manner as the *cleanowned* tool) so that they'll reequip with replacement clothes.

If there are not enough clothes available, manager orders will be generated to manufacture some more. `tailor` will intelligently create orders using raw materials that you have on hand in the fort. For example, if you have lots of silk, but no cloth, then `tailor` will order only silk clothing to be made.

Usage

```
enable tailor
tailor status
tailor materials <material> [<material> ...]
```

By default, `tailor` will prefer using materials in this order:

```
silk cloth yarn leather
```

but you can use the `tailor materials` command to restrict which materials are used, and in what order.

Example

`tailor materials silk cloth yarn` Restrict the materials used for automatically manufacturing clothing to silk, cloth, and yarn, preferred in that order. This saves leather for other uses, like making armor.

4.6.230 tame

Tags: fort | armok | animals

Command: `tame`

Tame and train animals.

Instantly set the training level of the selected animal or tame them completely.

Usage

```
tame --read
tame --set <level>
```

Examples

`tame --read` Show the training level for the selected animal.

`tame --set 7` Tame the selected animal.

`tame --set 8` Make the selected animal revert to a wild state.

Levels

The level numbers have the following definitions:

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

4.6.231 teleport

Tags: fort | armok | units

Command: `teleport`

Teleport a unit anywhere.

This tool teleports any unit, friendly or hostile, to somewhere else on the map.

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

Usage

```
teleport [--unit <id>] [-x <x> -y <y> -z <z>]
```

When teleporting, if no unit id is specified, the unit under the cursor is used. If no coordinates are specified, then the coordinates under the cursor are used. Either the unit id or the coordinates must be specified for this command to be useful.

You can use the *cprobe* command to discover a unit's id, or the *position* command to discover the map coordinates under the cursor.

Examples

Discover the id of the unit beneath the cursor and then teleport that unit to a new cursor position:

```
cprobe
teleport --unit 2342
```

Discover the coordinates under the cursor, then teleport a selected unit to that position:

```
position
teleport -x 34 -y 20 -z 163
```

Teleport unit `1234` to `56,115,26`:

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

4.6.232 tidlers

Tags: interface

Command: `tidlers`

Change where the idlers count is displayed.

This tool simply cycles the idlers count among the possible positions where the idlers count can be placed, including making it disappear entirely.

Usage

```
tidlers
```

4.6.233 tiletypes

Tags: adventure | fort | armok | map

Command: `tiletypes`

Paints tiles of specified types onto the map.

Command: `tiletypes-command`

Run tiletypes commands.

Command: `tiletypes-here`

Paint map tiles starting from the cursor.

Command: `tiletypes-here-point`

Paint the map tile under the cursor.

You can use the *probe* command to discover properties of existing tiles that you'd like to copy. If you accidentally paint over a vein that you want back, *fixveins* may help.

The tool works with a brush, a filter, and a paint specification. The brush determines the shape of the area to affect, the filter selects which tiles to affect, and the paint specification determines how to affect those tiles.

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

Usage

`tiletypes` Start the interactive terminal prompt where you can iteratively modify the brush, filter, and paint specification and get help on syntax elements. When in the interactive prompt, type `quit` to get out.

`tiletypes-command <command> [; <command> ...]` Run `tiletypes` commands from outside the interactive prompt. You can use this form from hotkeys or *dfhack-run* to set specific tiletypes properties. You can run multiple commands on one line by separating them with ` ; ` – that's a semicolon with a space on either side. See the *Commands* section below for an overview of commands you can run.

`tiletypes-here [<options>]` Apply the current options set in `tiletypes` and/or `tiletypes-command` at the in-game cursor position, including the brush. Can be used from a hotkey.

`tiletypes-here-point [<options>]` Apply the current options set in `tiletypes` and/or `tiletypes-command` at the in-game cursor position to a single tile (ignoring brush settings). Can be used from a hotkey.

Examples

`tiletypes-command filter material STONE ; f shape WALL ; paint shape FLOOR` Turn all stone walls into floors, preserving the material.

`tiletypes-command p any ; p s wall ; p sp normal` Clear the paint specification and set it to unsmoothed walls.

`tiletypes-command f any ; p stone marble ; p sh wall ; p sp normal ; r 10 10` Prepare to paint a 10x10 area of marble walls, ready for harvesting for flux.

`tiletypes-command f any ; f designated 1 ; p any ; p hidden 0 ; block ; run` Set the filter to match designated tiles, the paint specification to unhide them, and the brush to cover all tiles in the current block. Then run itThis is useful for unhiding tiles you wish to dig out of an aquifer so the game doesn't pause and undesignate adjacent tiles every time a new damp tile is "discovered".

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.

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

Commands

Commands can set the brush or modify the filter or paint options. When at the interactive `tiletypes>` prompt, the command `run` (or hitting enter on an empty line) will apply the current filter and paint specification with the current brush at the current cursor position. The command `quit` will exit.

Brush commands

p, point Use the point brush.

r, range <width> <height> [<depth>] Use the range brush with the specified width, height, and depth. If not specified, depth is 1, meaning just the current z-level. The range starts at the position of the cursor and goes to the east, south and up (towards the sky).

block Use the block brush, which includes all tiles in the 16x16 block that includes the cursor.

column Use the column brush, which ranges from the current cursor position to the first solid tile above it. This is useful for filling the empty space in a cavern.

Filter and paint commands

The general forms for modifying the filter or paint specification are:

f, filter <options> Modify the filter.

p, paint <options> Modify the paint specification.

The options identify the property of the tile and the value of that property:

any Reset to default (no filter/paint).

s, sh, shape <shape> Tile shape information. Run `:lua @df.tiletype_shape` to see valid shapes, or use a shape of `any` to clear the current setting.

m, mat, material <material> Tile material information. Run `:lua @df.tiletype_material` to see valid materials, or use a material of `any` to clear the current setting.

sp, special <special> Tile special information. Run `:lua @df.tiletype_special` to see valid special values, or use a special value of `any` to clear the current setting.

v, var, variant <variant> Tile variant information. Run `:lua @df.tiletype_variant` to see valid variant values, or use a variant value of `any` to clear the current setting.

a, all [<shape>] [<material>] [<special>] [<variant>] Set values for any or all of shape, material, special, and/or variant, in any order.

d, designated 0|1 Only useful for the filter, since you can't "paint" designations.

h, hidden 0|1 Whether a tile is hidden. A value of `0` means "revealed".

l, light 0|1 Whether a tile is marked as "Light". A value of `0` means "dark".

st, subterranean 0|1 Whether a tile is marked as "Subterranean".

sv, skyview 0|1 Whether a tile is marked as "Outside". A value of `0` means "inside".

aqua, aquifer 0|1 Whether a tile is marked as an aquifer.

stone <stone type> Set a particular type of stone, creating veins as required. To see a list of valid stone types, run: `:lua for _,mat in ipairs(df.global.world.raws.inorganics) do if mat.material.flags.IS_STONE and not mat.material.flags.NO_STONE_STOCKPILE then print(mat.id) end end` Note that this command paints under ice and constructions, instead of overwriting them. Also note that specifying a specific `stone` will cancel out anything you have specified for `material`, and vice-versa.

veintype <vein type> Set a particular vein type for the `stone` option to take advantage of the different boulder drop rates. To see valid vein types, run `:lua @df.inclusion_type`, or use vein type `CLUSTER` to reset to the default.

4.6.234 timestream

Tags: fort | fps

Command: `timestream`

Fix FPS death.

Do you remember when you first start a new fort, your initial 7 dwarves zip around the screen and get things done so quickly? As a player, you never had to wait for your initial dwarves to move across the map. Don't you wish that your fort of 200 dwarves could be as zippy? This tool can help.

`timestream` keeps the game running quickly by dynamically adjusting the calendar speed relative to the frames per second that your computer can support. Your dwarves spend the same amount of in-game time to do their tasks, but the time that you, the player, have to wait for the dwarves to do things speeds up. This means that the dwarves in your fully developed fort appears as energetic as a newly created one, and mature forts are much more fun to play.

If you just want to change the game calendar speed without adjusting dwarf speed, this tool can do that too. Your dwarves will just be able to get less/more done per season (depending on whether you speed up or slow down the calendar).

Usage

timestream --units [--fps <target FPS>] Keep the game running as responsively as it did when it was running at the given frames per second. Dwarves get the same amount done per game day, but game days go by faster. If a target FPS is not given, it defaults to 100.

timestream --rate <rate>, timestream --fps <target FPS> Just change the rate of the calendar, without corresponding adjustments to units. Game responsiveness will not change, but dwarves will be able to get more (or less) done per game day. A rate of 1 is "normal" calendar speed. Alternately, you can run the calendar at a rate that it would have moved at while the game was running at the specified frames per second.

Examples

`timestream --units` Keep the game running as quickly and smoothly as it did when it ran "naturally" at 100 FPS. This mode makes things much more pleasant for the player without giving any advantage/disadvantage to your in-game dwarves.

`timestream --rate 2` Calendar runs at 2x normal speed and units get half as much done as usual per game day.

`timestream --fps 100` Calendar runs at a dynamic speed to simulate 100 FPS. Units get a varying amount of work done per game day, but will get less and less done as your fort grows and your unadjusted FPS decreases.

`timestream --rate 1` Reset everything back to normal.

4.6.235 title-folder

Tags: interface

Displays the DF folder name in the window title bar.

Usage

```
enable title-folder
```

4.6.236 title-version

Tags: interface

Displays the DFHack version on DF's title screen.

Usage

```
enable title-version
```

4.6.237 trackstop

Tags: fort | gameplay | buildings

Add dynamic configuration options for track stops.

When enabled, this plugin adds a `q` menu for track stops, which is completely blank in vanilla DF. 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.

Usage

```
enable trackstop
```

4.6.238 troubleshoot-item

Tags: fort | inspection | items

Command: `troubleshoot-item`

Inspect properties of the selected item.

This tool lets you inspect internal properties of the selected item. This is useful for troubleshooting issues such as dwarves refusing to touch certain items.

Usage

```
troubleshoot-item
```

4.6.239 tubefill

Tags: fort | armok | map

Command: `tubefill`

Replenishes mined-out adamantine.

Veins that were originally hollow will be left alone.

Usage

```
tubefill [hollow]
```

Specify `hollow` to fill in naturally hollow veins too, but be aware that this will trigger a demon invasion on top of your miner when you dig into the region that used to be hollow. You have been warned!

4.6.240 twaterlvl

Tags: interface

Command: `twaterlvl`

Show/hide numeric liquid depth on the map.

Keybinding: CtrlW in `dwarfmode|dungeonmode`

You can use this tool to toggle between displaying/not displaying liquid depth as numbers on the map.

Usage

```
twaterlvl
```

4.6.241 tweak

Tags: adventure | fort | armok | bugfix | fps | interface

Command: `tweak`

A collection of tweaks and bugfixes.

Usage

```
tweak <command> [disable]
```

Run the `tweak` command to run the tweak or enable its effects. For tweaks that have persistent effects, append the `disable` keyword to disable them.

One-shot commands:

`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).

`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 labor management programs (like *manipulator* or Dwarf Therapist) 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.

Commands that persist until disabled or DF quits:

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

`advmode-contained` Fixes custom reactions with container inputs in advmode (Bug 6202) 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 in vanilla DF (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 condition material list (Bug 9905).

`craft-age-wear` Fixes crafted items not 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` 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 `tweak stable-temp` to stop updates again when equilibrium is disturbed.

`fast-trade` Makes Shift-Down in the Move Goods to Depot and Trade screens toggle the current item (fully, in case of a stack), and scroll down one line. Shift-Up undoes the last Shift-Down by scrolling up one line and then toggle the item.

`fps-min` Fixes the in-game minimum FPS setting (Bug 6277).

`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` When meeting with a liaison, makes Shift+Left/Right arrow adjust all items in 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` Makes built nestboxes use the color of their material.

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

`pausing-fps-counter` Replace fortress mode FPS counter with one that stops counting when paused.

`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.

4.6.242 type

Tags: dfhack

Command: `type`

Describe how a command is implemented.

DFHack commands can be provided by plugins, scripts, or by the core library itself. The `type` command can tell you which is the source of a particular command.

Usage

```
type <command>
```

4.6.243 undump-buildings

Tags: fort | productivity | buildings

Command: `undump-buildings`

Undesignate building base materials for dumping.

If you designate a bunch of tiles in dump mode, all the items on those tiles will be marked for dumping. Unfortunately, if there are buildings on any of those tiles, the items that were used to *build* those buildings will also be uselessly and confusingly marked for dumping.

This tool will scan for buildings that have their construction materials marked for dumping and will unmark them.

Usage

```
undump-buildings
```

4.6.244 unforbid

Tags: fort | productivity | items

Command: `unforbid`

Unforbid all items.

This tool quickly and easily unforbids all items. This is especially useful after a siege to allow cleaning up the mess (or dumping of caged prisoner's equipment with *stripcaged*).

Usage

```
unforbid all [<options>]
```

Options

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

4.6.245 ungeld

Tags: fort | armok | animals

Command: `ungeld`

Undo gelding for an animal.

This tool will restore an animal's ability to reproduce after it has been gelded. Also see *geld* if you'd like to re-geld the animal.

Usage

```
ungeld [--unit <id>]
```

Options

`--unit <id>` Ungelds the unit with the specified ID. If this option is not specified, the default is to use the currently selected unit.

4.6.246 uniform-unstick

Tags: fort | bugfix | military

Command: `uniform-unstick`

Make military units reevaluate their uniforms.

This tool prompts military units to reevaluate their uniform, making them remove and drop 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).

Uniforms that have no issues are being properly worn will not be affected.

Usage

`uniform-unstick [--all]` List problems with the uniform for the currently selected unit (or all units).

`uniform-unstick [--all] <strategy options>` Fix the problems with the unit's uniform (or all units' uniforms) using the specified strategies.

Examples

`uniform-unstick --all --drop --free` Fix all issues with uniforms that have only one item per body part (like all default uniforms).

Strategy options

`--drop` Force the unit to drop conflicting worn items onto the ground, where they can then be reclaimed in the correct order.

`--free` Remove to-equip items from containers or other's inventories and place them on the ground, ready to be claimed. This is most useful when someone else is wearing/holding the required items.

4.6.247 unload

Tags: dfhack

Command: `unload`

Unload a plugin from memory.

Also see *load* and *reload* for related actions.

Usage

```
unload <plugin> [<plugin> ...]
unload -a|--all
```

You can unload individual named plugins or all plugins at once.

4.6.248 unretire-anyone

Tags: adventure | embark | armok

Command: `unretire-anyone`

Adventure as any living historical figure.

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

To use, simply run the command 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, the same as when regaining control of a retired adventurer.

Usage

```
unretire-anyone
```

Options

-d, --dead Enables user to unretire a dead historical figure to play as in adventure mode. For instance, a user may wish to unretire and then play as a particular megabeast that had died during world-gen.

4.6.249 unsuspend

Tags: fort | productivity | jobs

Command: `unsuspend`

Unsuspends building construction jobs.

Unsuspends building construction jobs, except for jobs managed by *buildingplan* and those where water flow is greater than 1. This allows you to quickly recover if a bunch of jobs were suspended due to the workers getting scared off by wildlife or items temporarily blocking building sites.

See *autounsuspend* for periodic automatic unsuspending of suspended jobs.

Usage

```
unsuspend
```

Overlay

This script also provides an overlay that is managed by the *overlay* framework. When enabled, it will display a colored 'X' over suspended buildings. A green 'X' indicates that the building is waiting on materials, and *buildingplan* will unsuspend it for you when those materials become available. A yellow 'X' means that the building is suspended and that you can unsuspend it manually or with the *unsuspend* command. A red 'X' indicates that the building has been re-suspended multiple times, and that you might need to look into whatever is preventing the building from being built.

4.6.250 view-item-info

Tags: adventure | fort | interface

Command: `view-item-info`

Extend item and unit descriptions with more information.

This tool extends the item or unit description 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.

Usage

```
enable view-item-info
```

Info for modded items

The associated `scripts/internal/view-item-info/item-descriptions` script supplies custom descriptions of items. Mods can extend or override the descriptions in that file by supplying a similarly formatted file named `raw/scripts/more-item-descriptions.lua`. Both work as sparse lists, so missing items simply go undescribed if not defined in the fallback.

4.6.251 view-unit-reports

Tags: fort | inspection | military

Command: `view-unit-reports`

Show combat reports for a unit.

Keybinding: CtrlShiftR in `dwarfmode|unit|unitlist|joblist|dungeon_monsterstatus|layer_unit_relationship|item`

Show combat reports specifically for the selected unit. You can select a unit with the cursor in `v` mode, from the list in `u` mode, or from the unit/corpse/splatter list in `k` mode. You can also select the newest unit with a particular race when looking at a race-specific blood spatter in `k` mode.

Usage

```
view-unit-reports
```

4.6.252 warn-starving

Tags: fort | animals | units

Command: `warn-starving`

Report units that are dangerously hungry, thirsty, or drowsy.

If any (live) units are starving, very thirsty, or very drowsy, the game will pause and you'll get a warning dialog telling you which units are in danger. This gives you a chance to rescue them (or take them out of their cages) before they die.

Usage

```
warn-starving [all] [sane]
```

Examples

`warn-starving all sane` Report on all currently distressed units, excluding insane units that you wouldn't be able to save anyway.

`repeat --time 10 --timeUnits days --command [ warn-starving sane ]` Every 10 days, report any (sane) distressed units that haven't already been reported.

Options

`all` Report on all distressed units, even if they have already been reported. By default, only newly distressed units that haven't already been reported are listed.

`sane` Ignore insane units.

4.6.253 warn-stealers

Tags: fort | armok | auto | units

Command: `warn-stealers`

Watch for and warn about units that like to steal your stuff.

This script will watch for new units entering the map and will make a zoomable announcement whenever a creature that can eat food, guzzle drinks, or steal items moves into a revealed location. It will also re-warn about all revealed stealers when enabled.

Usage

```
enable warn-stealers
```

4.6.254 weather

Tags: fort | armok | inspection | map

Command: `weather`

Change the weather.

This tool allows you to inspect or control the weather.

Usage

`weather` Print a map of the local weather.

`weather clear|rain|snow` Change the weather as specified.

Examples

`weather clear` Make it stop raining/snowing.

4.6.255 workNow

Tags: fort | auto | labors

Command: `workNow`

Reduce the time that dwarves idle after completing a job.

After finishing a job, dwarves will wander away for a while before picking up a new job. This plugin will automatically poke the game to assign dwarves to new tasks.

Usage

`workNow` Print current plugin status.

`workNow 0` Stop monitoring and poking.

`workNow 1` Poke the game to assign dwarves to tasks whenever the game is paused.

`workNow 2` Poke the game to assign dwarves to tasks whenever a dwarf finishes a job.

4.6.256 workflow

Tags: fort | auto | jobs

Command: `workflow`

Manage automated item production rules.

Command: `fix-job-postings`

Fixes crashes caused by old versions of workflow.

Manage repeat jobs according to stock levels. *gui/workflow* provides a simple front-end integrated in the game UI.

When the plugin is enabled, it protects all repeat jobs from removal. If they do disappear due to any cause (raw materials not available, manual removal by the player, etc.), they are immediately re-added to their workshop and suspended.

If 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.

There is a good amount of overlap between this plugin and the vanilla manager workorders, and both systems have their advantages. Vanilla manager workorders can be more expressive about when to enqueue jobs. For example, you can gate the activation of a vanilla workorder based on availability of raw materials, which you cannot do in `workflow`. However, `workflow` is often more convenient for quickly keeping a small stock of various items on hand without having to configure all the vanilla manager options. Also see the *orders* plugin for a library of manager orders that may make managing your stocks even more convenient than `workflow` can.

Usage

`enable workflow` Start monitoring for and managing workshop jobs that are set to repeat.

`workflow enable|disable drybuckets` Enables/disables automatic emptying of abandoned water buckets.

`workflow enable|disable auto-melt` Enables/disables automatic resumption of repeat melt jobs when there are objects to melt.

`workflow count <constraint-spec> <target> [gap]` Set a constraint, counting every stack as 1 item. If a gap is specified, stocks are allowed to dip that many items below the target before relevant jobs are resumed.

`workflow amount <constraint-spec> <target> [gap]` Set a constraint, counting all items within stacks. If a gap is specified, stocks are allowed to dip that many items below the target before relevant jobs are resumed.

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

`workflow unlimit-all` Delete all constraints.

`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.

`fix-job-postings [dry-run]` Fixes crashes caused by the version of workflow released with DFHack 0.40.24-r4. It will be run automatically if needed. If your save has never been run with this version, you will never need this command. Specify the `dry-run` keyword to see what this command would do without making any changes to game state.

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
```

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 for 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 (`LOCAL`) or items below a certain quality (1-5, with 5 being masterwork).

4.6.257 workorder

Tags: fort | productivity | workorders

Command: `workorder`

Create manager workorders.

This tool can enqueue work orders as if you were using the `j-m-q` interface. It also has some convenience functions, such as automatically counting how many creatures can be milked or sheared for `MilkCreature` or `ShearCreature` jobs. It can also take existing orders into account to ensure that the quantity produced by *all* enqueued workorders for a specified job type totals to a specified amount.

Usage

`workorder -l <filter>`, `workorder --listtypes <filter>` Print all values for relevant DF types (`job_type`, `item_type` etc.) that will be useful for assembling the workorder json. You can pass a filter to only print types that match a pattern.

`workorder <jobtype> [<amount>]` The job type is the number or name from `df.job_type` and the amount is the quantity for the generated workorder. The amount can be omitted for `MilkCreature` and `ShearCreature` jobs, and `workorder` will scan your pets for milkable or shearable creatures and fill the correct number in. Note that this syntax cannot specify the material of the item produced by the job. If you need more specificity, you can describe the job in JSON format (see the next two command forms).

`workorder <json>` Create a workorder whose properties are specified in the given JSON. See below for examples and the complete format specification.

`workorder --file <filename>` Loads the json representation of a workorder from the specified file in `dfhack-config/workorder/`.

Examples

`workorder MakeCharcoal 100` Enqueue a workorder to make 100 bars of charcoal.

`workorder MakeTable 10` Enqueue a workorder to make 10 tables of unspecified material. The material will be determined by which workshop ends up picking up the job.

`repeat --name autoShearCreature --time 14 --timeUnits days --command [ workorder ShearCreature ]`
Automatically shear any pets that are ready to be sheared.

`repeat --name autoMilkCreature --time 14 --timeUnits days --command [ workorder "{\"job\":\"MilkCreatur`
Automatically milk any pets that are ready to be milked (but only if there are at least 5 empty buckets available to receive the milk).

`workorder "{\"job\":\"EncrustWithGems\",\"item_category\":[\"finished_goods\"],\"amount_total\":5}"`
Add an order to `EncrustWithGems` five `finished_goods` using any material (since a material is not specified).

JSON string specification

The JSON representation of a workorder must be a valid Lua string literal (note usage of \ in the JSON examples above). You can export existing manager orders with the *orders* command and look at the created `.json` file in `dfhack-config/orders` to see how a particular order can be represented.

Note that, unlike *orders*, `workorder` is meant for dynamically creating new orders, so even if fields like `amount_left`, `is_active` or `is_validated` are specified in the JSON, they will be ignored in the generated orders.

Also:

- You only need to fill in `id` if it is used for order conditions
- If `frequency` is unspecified, it defaults to `OneTime`
- The `amount_total` field can be missing (only valid for `MilkCreature` or `ShearCreature` jobs) or it can be raw Lua code called as `load(code)(order, orders)` that must return an integer.

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

4.6.258 workorder-recheck

Tags: fort | workorders

Command: `workorder-recheck`

Recheck start conditions for a manager workorder.

Keybinding: AltR in `jobmanagement/Main`

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. This is especially useful for an order that had its conditions met when it was started, but the requisite items have since disappeared and the workorder is now generating job cancellation spam.

Usage

```
workorder-recheck
```

4.6.259 xlsxreader

Tags: dev

Provides a Lua API for reading xlsx files.

See *xlsxreader* for details.

4.6.260 zone

Tags: fort | productivity | animals | buildings

Command: `zone`

Manage activity zones, cages, and the animals therein.

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

Usage

`enable zone` Add helpful filters to the pen/pasture sidebar menu (e.g. show only caged grazers).

`zone set` Set zone or cage under cursor as default for future `assign` or `tocages` commands.

`zone assign [<zone id>] [<filter>]` Assign unit(s) to the zone with the given ID, or to the most recent pen or pit marked with the `set` command. If no filters are set, then a unit must be selected in the in-game ui.

`zone unassign [<filter>]` Unassign selected creature from its zone.

`zone nick <nickname> [<filter>]` Assign the given nickname to the selected animal or the animals matched by the given filter.

`zone remnick [<filter>]` Remove nicknames from the selected animal or the animals matched by the given filter.

`zone enumnick <nickname prefix> [<filter>]` Assign enumerated nicknames (e.g. "Hen 1", "Hen 2"…).

`zone tocages [<filter>]` Assign unit(s) to cages that have been built inside the pasture selected with the `set` command.

`zone uinfo [<filter>]` Print info about unit(s). If no filters are set, then a unit must be selected in the in-game ui.

`zone zinfo` Print info about the zone(s) and any buildings under the cursor.

Examples

Before any `assign` or `tocages` examples can be used, you must first move the cursor over a pen/pasture or pit zone and run `zone set` to select the zone.

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

`zone assign all own caged grazer nick ineedgrass` Assign all of your 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 of your animals who are not grazers (excluding cats) to the selected pasture. " zone assign all own milkable not grazern"

`zone assign all own female milkable not grazer` Assign all of your non-grazing milkable creatures to the selected pasture or cage.

`zone assign all own race DWARF maxage 2` Throw all useless kids into a pit :) They'll be fine I'm sure.

`zone nick donttouchme` Nicknames all units in the current default zone or cage to 'donttouchme'. This is especially useful for protecting a group of animals assigned to a pasture or cage from being "processed" by *autobutcher*.

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

Filters

all Process all units.

count <n> Process only up to n units.

unassigned Not assigned to zone, chain or built cage.

minage <years> Minimum age. Must be followed by a number.

maxage <years> Maximum age. Must be followed by a number.

not Negates the next filter keyword. All of the keywords documented below are negatable.

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

caged In a built cage.

own From own civilization. You'll usually want to include this filter.

war Trained war creature.

hunting Trained hunting creature.

tamed Creature is tame.

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`.

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

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

male Creature is male.

female Creature is female.

egglayer Race lays eggs. If you want units who actually lay eggs, also specify `female`.

grazer Race is a grazer.

milkable Race is milkable. If you want units who actually can be milked, also specify `female`.

merchant Is a merchant / belongs to a merchant. Should only be used for pitting or slaughtering, not for stealing animals.

Usage with single units

One convenient way to use the zone tool is to bind the commands `zone assign` and `zone set` to hotkeys. Place the in-game cursor over a pen/pasture or pit and use the `zone set` hotkey 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 the `zone assign` hotkey to assign them to their new home. Allows pitting your own dwarves, by the way.

Matching with filters

All filters can be used together with the `assign` and `tocages` commands.

Note that it's not possible to reassign units who are inside built cages or chained, though this likely won't matter because if you have gone to the trouble of creating a zoo or chaining a creature, you probably wouldn't want them reassigned anyways. Also, `zone` will avoid caging owned pets because the owner uncages them after a while which results in infinite hauling back and forth.

Most filters should include an `own` element (which implies `tame`) unless you want to use `zone assign` for pitting hostiles. The `own` filter ignores dwarves unless you explicitly specify `race DWARF` (so it's safe to use `assign all own` to one big pasture if you want to have all your animals in the same place).

The `egglayer` and `milkable` filters should be used together with `female` unless you want the males of the race included. 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 likely further optional filters) it will rename units matching the filter conditions.

Cage zones

The `tocages` command assigns units to a set of cages, for example a room next to your butcher shop(s). Units 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 run `zone tocages <filter>`. `tocages` can be used together with `nick` or `remnick` to adjust nicknames while assigning to cages.

4.6.261 devel/all-bob

Tags: dev

Command: `devel/all-bob`

Changes the first name of all units to "Bob"..

Useful for testing *modtools/interaction-trigger* events.

Usage

```
devel/all-bob
```

4.6.262 devel/annc-monitor

Tags: dev

Command: `devel/annc-monitor`

Track announcements and reports and echo them to the console.

This tool monitors announcements and reports and echoes their contents to the console.

Usage

```
enable devel/annc-monitor
devel/annc-monitor report enable|disable
```

Combat report monitoring is disabled by default.

4.6.263 devel/block-borders

Tags: dev | map

Command: `devel/block-borders`

Outline map blocks on the map screen.

This tool displays an overlay that highlights the borders of map blocks. See *Maps API* for details on map blocks.

Usage

```
devel/block-borders
```

4.6.264 devel/check-other-ids

Tags: dev

Command: `devel/check-other-ids`

Verify that game entities are referenced by the correct vectors.

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.

Usage

```
devel/check-other-ids
```

4.6.265 devel/check-release

Tags: dev

Command: `devel/check-release`

Perform basic checks for DFHack release readiness.

This script is run as part of the DFHack release process to check that release flags are properly set.

Usage

```
devel/check-release
```

4.6.266 devel/clear-script-env

Tags: dev

Command: `devel/clear-script-env`

Clear a lua script environment.

This tool can clear the environment of the specified lua script(s). This is useful during development since if you remove a global function, an old version of the function will stick around in the environment until it is cleared.

Usage

```
devel/clear-script-env <script name> [<script name> ...]
```

Example

`devel/clear-script-env gui/quickfort` Clear the *gui/quickfort* global environment, resetting state that normally persists from run to run.

4.6.267 devel/click-monitor

Tags: dev

Command: `devel/click-monitor`

Displays the grid coordinates of mouse clicks in the console.

This tool will watch for mouse activity and print relevant information to the console. Useful for plugin/script development.

Usage

:: enable devel/click-monitor

4.6.268 devel/cmptiles

Tags: dev | inspection

Command: `devel/cmptiles`

List or compare two tiletype material groups.

Lists and/or compares two tiletype material groups. You can see the list of valid material groups by running:

```
:lua @df.tiletype_material
```

Usage

```
devel/cmptiles material1 [material2]
```

4.6.269 devel/dump-offsets

Tags: dev

Command: `devel/dump-offsets`

Dump the contents of the table of global addresses.

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 script dumps the contents of the table of global addresses (new in 0.44.01).

Usage

```
devel/dump-offsets all|<global var>
```

Passing global names as arguments calls `setAddress()` to set those globals' addresses in-game. Passing "all" does this for all globals.

4.6.270 devel/dump-rpc

Tags: dev

Command: `devel/dump-rpc`

Dump RPC endpoint info.

Write RPC endpoint information to the specified file.

Usage

```
devel/dump-rpc <filename>
```

4.6.271 devel/eventful-client

Tags: dev

Command: `devel/eventful-client`

Simple client for testing event callbacks.

You can use this tool to discover when specific events fire and to test when callbacks are called for different callback frequency settings.

Note this script does not handle the eventful reaction or workshop events.

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

4.6.272 devel/export-dt-ini

Tags: dev

Command: `devel/export-dt-ini`

Export memory addresses for Dwarf Therapist configuration.

This tool exports an ini file containing memory addresses for Dwarf Therapist.

Usage

```
devel/export-dt-ini
```

4.6.273 devel/find-offsets

Tags: dev

Command: `devel/find-offsets`

Find memory offsets of DF data structures.

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.

To find the first few globals, you must run this script 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*.

Usage

```
devel/find-offsets all|<global names> [nofeed] [nozoom]
```

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

4.6.274 devel/find-primitive

Tags: dev

Command: `devel/find-primitive`

Discover memory offsets for new variables.

This tool helps find a primitive variable in DF's data section, relying on the user to change its value and then scanning for memory that has changed to that new 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.

Run `devel/find-primitive help` for a list of valid data types.

4.6.275 devel/find-twbt

Tags: dev

Command: `devel/find-twbt`

Display the memory offsets of some important TWBT functions.

Finds some TWBT-related offsets - currently just `twbt_render_map`.

Usage

```
devel/find-twbt
```

4.6.276 devel/hello-world

Tags: dev

Command: `devel/hello-world`

A basic GUI example script.

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

Usage

```
devel/hello-world
```

4.6.277 devel/inject-raws

Tags: dev

Command: `devel/inject-raws`

Add objects and reactions into an existing world.

WARNING: THIS SCRIPT CAN PERMANENTLY 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 then simply saving and reloading the game.

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

Usage

```
devel/inject-raws reaction|<building type>|<item type> TOKEN [TOKEN ...]
```

Example

```
devel/inject-raws trapcomp ITEM_TRAPCOMP_STEAM_PISTON workshop STEAM_ENGINE MAGMA_STEAM_
↪ENGINE reaction STOKE_BOILER
```

4.6.278 devel/inspect-screen

Tags: dev

Command: `devel/inspect-screen`

Show glyph and color info for screen tiles.

This script pops up a panel that shows the displayed character for the selected tile and foreground/background color information. You can move the cursor around the screen to select the tile you want information on.

Note that this script does not work if you are using TWBT.

Usage

```
devel/inspect-screen
```

4.6.279 devel/kill-hf

Tags: dev

Command: `devel/kill-hf`

Kill a historical figure.

This tool can kill the specified historical figure, even if off-site, or terminate a pregnancy. Useful for working around Bug 11549.

Usage

```
devel/kill-hf [-p|--pregnancy] [-n|--dry-run] <histfig_id>
```

Options

`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 instead of making any changes.

4.6.280 devel/light

Tags: dev | graphics

Command: `devel/light`

Experiment with lighting overlays.

This is an experimental lighting engine for DF, using the *rendermax* plugin.

Press ~ to recalculate lighting. Press ` to exit.

Usage

```
devel/light [static]
```

Pass `static` to not recalculate lighting when in game.

4.6.281 devel/list-filters

Tags: dev

Command: `devel/list-filters`

List input items for the selected building type.

This tool lists input items for the building that is currently being built. You must be in build mode and have a building type selected for placement. This is where the filters in `lua/dfhack/buildings.lua` come from.

Usage

```
devel/list-filters
```

4.6.282 devel/lsmem

Tags: dev

Command: `devel/lsmem`

Print memory ranges of the DF process.

Useful for checking whether a pointer is valid, whether a certain library/plugin is loaded, etc.

Usage

```
devel/lsmem
```

4.6.283 devel/lua-example

Tags: dev

Command: `devel/lua-example`

An example lua script.

This is an example Lua script which just reports the number of times it has been called. Useful for testing environment persistence.

4.6.284 devel/luacov

Tags: dev

Command: `devel/luacov`

Lua script coverage report generator.

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.

Usage

```
luacov [options] [pattern...]
```

Examples

`devel/luacov` Report on all DFHack lua scripts.

`devel/luacov -c quickfort` Report only on quickfort source files and then clear the stats. This is useful to run between test runs to see the coverage of your test changes.

`devel/luacov quickfort hack/lua` Report only on quickfort and DFHack library lua source files.

Options

`-c`, `--clear` Remove accumulated metrics after generating the report, ensuring the next report starts from a clean slate.

4.6.285 devel/modstate-monitor

Tags: dev

Command: `devel/modstate-monitor`

Display changes in key modifier state.

This tool will show changes in `Ctrl`, `Alt`, and `Shift` modifier states.

Usage

```
enable devel/modstate-monitor
```

4.6.286 devel/nuke-items

Tags: dev | fps | items

Command: `devel/nuke-items`

Deletes all free items in the game.

This tool deletes **ALL** items not referred to by units, buildings, or jobs. Intended solely for lag investigation.

Usage

```
devel/nuke-items
```

4.6.287 devel/pop-screen

Tags: dev | interface

Command: `devel/pop-screen`

Forcibly closes the current screen.

Running this tool 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`)

Usage

```
devel/pop-screen
```

4.6.288 devel/prepare-save

Tags: dev

Command: `devel/prepare-save`

Set internal game state to known values for memory analysis.

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.

Usage

```
devel/prepare-save
```

4.6.289 devel/print-args

Tags: dev

Command: `devel/print-args`

Echo parameters to the output.

Prints all the arguments you supply to the script, one per line.

Usage

```
devel/print-args [<args>]
```

4.6.290 devel/print-args2

Tags: dev

Command: `devel/print-args2`

Echo parameters to the output.

Prints all the arguments you supply to the script, one per line, with quotes around them.

Usage

```
devel/print-args2 [<args>]
```

4.6.291 devel/print-event

Tags: dev | inspection

Command: `devel/print-event`

Show historical events.

This tool displays the description of a historical event.

Usage

devel/print-event --index <num> Displays the historical event at the given index.

devel/print-event --id <id> Displays the historical event by the given id.

4.6.292 devel/query

Tags: dev | inspection

Command: `devel/query`

Search/print data algorithmically.

Query is a useful script for finding and reading values of data structure fields. Players can use it to explore data structures or list elements of enums that they might need for another command. Developers can even integrate this script into another script to print data out for a player.

This script takes your data selection (e.g. a data table, unit, item, tile, etc.) and 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 fields that match a Lua pattern or set the value of specific fields.

Warning: This script searches 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. Set your limits wisely!

Tip: Should the need arise, you can stop `devel/query` from another shell with *kill-lua*, e.g. by running `dfhack-run kill-lua` from another terminal.

Usage

```
devel/query <source option> <query options> [<additional options>]
```

Examples

devel/query --unit --getfield id Prints the id of the selected unit.

devel/query --unit --search STRENGTH --maxdepth 3 Prints out information about the selected unit's `STRENGTH` attribute.

devel/query --unit --search physical_attrs --maxdepth 3 Prints out information about all of the selected unit's physical attributes.

devel/query --tile --search designation Prints out information about the selected tile's designation structure.

devel/query --tile --search "occup.*carv" Prints out information about the carving configuration for the selected tile.

`devel/query --table df --maxdepth 0` List the top-level fields in the `df` data structure.

`devel/query --table df.profession --findvalue FISH` Lists the enum values in the `df.profession` table that contain the substring `FISH`.

Source options

`--table <identifier>` Selects the specified table. You must use dot notation to denote sub-tables, e.g. `df.global.world`.

`--block` Selects the highlighted tile's block.

`--building` Selects the highlighted building.

`--item` Selects the highlighted item.

`--job` Selects the highlighted job.

`--plant` Selects the highlighted plant.

`--tile` Selects the highlighted tile's block, and then uses the tile's local position to index the 2D data.

`--unit` Selects the highlighted unit.

`--script <script>` Selects the specified script (which must support being included with *reqscript()*).

`--json <file>` Loads the specified json file as a table to query. The path starts at the DF root directory, e.g. `hack/scripts/dwarf_profiles.json`.

Query options

`--getfield <field>` Gets the specified field from the source.

`--search <pattern> [<pattern>]` Searches the source for field names with substrings matching any of the specified patterns.

`--findvalue <value>` Searches the source for field values matching the specified value.

`--maxdepth <value>` Limits the field recursion depth (default: 7).

`--maxlength <value>` Limits the number of items that the script will iterate through in a list (default: 2048).

`--excludetypes [a|bfnstu0]` Excludes native Lua data types. Single letters correspond to (in order): (a)ll types listed here, (b)oolean, (f)unction, (n)umber, (s)tring, (t)able, (u)serdata, nil values.

`--excludekinds [a|bces]` Excludes DF data types. Single letters correspond to (in order): (a)ll types listed here, (b)itfields, (c)lasses, (e)nums, (s)tructs.

`--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` Reduces output to one line (except with `--debugdata`) in cases where multiple lines of information is displayed for a field.

`--alignto <value>` Specifies the alignment column.

`--nopointers` Disables printing values which contain memory addresses.

`--debug <value>` Enables debug log verbosity for entries equal to or less than the value provided (valid values: 0-3).

`--debugdata` Prints type information under each field.

4.6.293 devel/save-version

Tags: dev | inspection

Command: `devel/save-version`

Display what DF version has handled the current save.

This tool displays the DF version that created the game, the most recent DF version that has loaded and saved the game, and the current DF version.

Usage

```
devel/save-version
```

4.6.294 devel/sc

Tags: dev

Command: `devel/sc`

Scan DF structures for errors.

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.

Usage

`devel/sc` Scan `world`.

`devel/sc -all` Scan all globals.

`devel/sc <expr>` Scan the result of the given expression.

4.6.295 devel/scanitemother

Tags: dev | inspection

Command: `devel/scanitemother`

Display the item lists that the selected item is part of.

When an item is selected in the UI, this tool will list the indices in `world.item.other[]` where the item appears. For example, if a piece of good meat is selected in the UI, this tool might output:

```
IN_PLAY
ANY_GOOD_FOOD
ANY_EDIBLE_RAW
ANY_EDIBLE_CARNIVORE
ANY_EDIBLE_BONECARN
ANY_EDIBLE_VERMIN
ANY_EDIBLE_VERMIN_BOX
ANY_CAN_ROT
ANY_COOKABLE
MEAT
```

Usage

```
devel/scanitemother
```

4.6.296 devel/send-key

Tags: dev | interface

Command: `devel/send-key`

Deliver key input to a viewscreen.

This tool can send a key to the current screen or to a parent screen. Note that 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> [<depth>]
```

The key to send is the name of an `interface_key` (see valid values by running `:lua @df.interface_key`, looking in `data/init/interface.txt`, or by checking `df.keybindings.xml` in the df-structures repository.

You can optionally specify the depth of the screen that you want to send the key to. `1` corresponds to the current screen's parent.

4.6.297 devel/spawn-unit-helper

Tags: dev | fort

Command: `devel/spawn-unit-helper`

Prepares the game for spawning creatures by switching to arena.

This script initializes game state to allow you to switch to arena mode, spawn creatures, and then switch back to fortress mode.

Usage

1. **Enter the k menu and change mode using** `rb_eval df.gametype = :DWARF_ARENA`
2. Spawn creatures with the normal arena mode UI (`c` ingame)
3. **Revert to forgress mode using** `rb_eval df.gametype = #{df.gametype.inspect}`
4. **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`

4.6.298 devel/test-perlin

Tags: dev

Command: `devel/test-perlin`

Generate an image based on perlin noise.

Generates an image using multiple octaves of perlin noise.

Usage

```
devel/test-perlin <fname.pgm> <density> <expr> [-xy <xyscale>] [-z <zscale>]
```

The inquisitive will have to inspect the code to determine the meaning of the parameters.

4.6.299 devel/unit-path

Tags: dev | inspection

Command: `devel/unit-path`

Inspect where a unit is going and how it's getting there.

When run with a unit selected, the path that the unit is currently following is highlighted on the map. You can jump between the unit and the destination tile.

Usage

```
devel/unit-path
```

4.6.300 devel/visualize-structure

Tags: dev | inspection | dfhack

Command: `devel/visualize-structure`

Display raw memory of a DF data structure.

Displays the raw memory of a structure, field by field. Useful for checking if structures are aligned.

Usage

```
devel/visualize-structure <lua expression>
```

Example

```
devel/visualize-structure df.global.cursor
```

4.6.301 devel/watch-minecarts

Tags: dev | inspection

Command: `devel/watch-minecarts`

Inspect minecart coordinates and speeds.

When running, this tool will log minecart coordinates and speeds to the console.

Usage

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

4.6.302 fix/blood-del

Tags: fort | bugfix

Command: `fix/blood-del`

Removes unusable liquids from caravan manifests.

This fix ensures future caravans won't bring barrels full of blood, ichor, or goo.

Usage

```
fix/blood-del
```

4.6.303 fix/corrupt-equipment

Tags: fort | bugfix | military

Command: `fix/corrupt-equipment`

Fixes some game crashes caused by corrupt military equipment.

This fix corrects some kinds of corruption that can occur in equipment lists, as in Bug 11014. Run this script at least every time a squad comes back from a raid.

Usage

```
fix/corrupt-equipment
```

Examples

`fix/corrupt-equipment` Run the fix manually a single time.

`repeat --time 100 --timeUnits ticks --command [ fix/corrupt-equipment ]` Automatically run the fix in the background every 100 ticks so you don't have to remember to run it manually.

Technical details

There are several types of corruption that have been 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.

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

4.6.304 fix/dead-units

Tags: fort | bugfix | units

Command: `fix/dead-units`

Remove dead units from the list so migrants can arrive again.

If so many units have died at your fort that your dead units list exceeds about 3000 units, migrant waves can stop coming. This fix removes uninteresting units (like slaughtered animals and nameless goblins) from the unit list, allowing migrants to start coming again.

Usage

```
fix/dead-units
```

4.6.305 fix/drop-webs

Tags: fort | bugfix | items

Command: `fix/drop-webs`

Make floating webs drop to the ground.

Webs can be left floating in mid-air for a variety of reasons, such as getting caught in a tree and the tree subsequently being chopped down (Bug 595). This tool finds the floating webs and makes them fall to the ground.

See *clear-webs* if you want to remove webs entirely.

Usage

`fix/drop-webs` Make webs that are floating in mid-air drop to the ground

`fix/drop-webs --all` Make webs that are above the ground for any reason (including being stuck in tree branches) fall to the ground.

4.6.306 fix/dry-buckets

Tags: fort | bugfix | items

Command: `fix/dry-buckets`

Allow discarded water buckets to be used again.

Sometimes, dwarves drop buckets of water on the ground if their water hauling job is interrupted. These buckets then become unavailable for any other kind of use, such as making lye. This tool finds those discarded buckets and removes the water from them. Buckets in wells or being actively carried or used by a job are not affected.

Usage

```
fix/dry-buckets
```

4.6.307 fix/item-occupancy

Tags: fort | bugfix | map

Command: `fix/item-occupancy`

Fixes errors with phantom items occupying site.

This tool diagnoses and fixes issues with nonexistent 'items occupying site', usually caused by hacking mishaps with items being improperly moved about.

Usage

```
fix/item-occupancy
```

Technical details

This tool 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

4.6.308 fix/loyaltycascade

Tags: fort | bugfix | units

Command: `fix/loyaltycascade`

Halts loyalty cascades where dwarves are fighting dwarves.

This tool aborts loyalty cascades by fixing units who consider their own civilization to be the enemy.

Usage

```
fix/loyaltycascade
```

4.6.309 fix/population-cap

Tags: fort | bugfix | units

Command: `fix/population-cap`

Ensure the population cap is respected.

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

The reason this tool is needed is that the game only updates the records of your current population when a dwarven caravan successfully leaves for the mountainhomes. If your population was under the cap at that point, you will continue to get migrant waves. If another caravan never comes (or is never able to leave), you'll get migrant waves forever.

This tool ensures the population value always reflects the current population of your fort.

Note that even with this tool, a migration wave can still overshoot the limit by 1-2 dwarves because the last migrant might choose to bring their family. Likewise, monarch arrival ignores the population cap.

Usage

```
fix/population-cap
```

Examples

`repeat --time 1 --timeUnits months --command [ fix/population-cap ]` Automatically run this fix after every migrant wave to keep the population values up to date.

4.6.310 fix/retrieve-units

Tags: fort | bugfix | units

Command: `fix/retrieve-units`

Allow stuck offscreen units to enter the map.

This script finds units that are marked as pending entry to the active map and forces them to enter. This can fix issues such as:

- 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.

Usage

```
fix/retrieve-units
```

4.6.311 fix/stable-temp

Tags: fort | bugfix | fps | map

Command: `fix/stable-temp`

Solve FPS issues caused by fluctuating temperature.

This tool instantly sets the temperature of all free-lying items to be in equilibrium with the environment. This effectively halts FPS-draining temperature updates until something changes, such as letting magma flow to new tiles.

To maintain this efficient state, use *tweak fast-heat*.

Usage

```
fix/stable-temp
```

4.6.312 fix/stuck-merchants

Tags: fort | bugfix | units

Command: `fix/stuck-merchants`

Dismiss merchants that are stuck off the edge of the map.

This tool dismisses merchants that haven't entered the map yet. This can fix Bug 9593. Where you get a trade caravan announcement, but no merchants ever enter the map.

This script should not be run if any merchants are on the map, so using it with *repeat* is not recommended.

Usage

`fix/stuck-merchants` Dismiss merchants that are stuck off the edge of the map.

`fix/stuck-merchants -n`, **`fix/stuck-merchants --dry-run`** List the merchants that would be dismissed, but make no changes.

4.6.313 fix/stuckdoors

Tags: fort | bugfix | buildings

Command: `fix/stuckdoors`

Allow doors that are stuck open to close.

This tool can fix doors that are stuck open due to incorrect map occupancy flags. If you see a door that never closes with no obvious cause, try running this tool.

Usage

```
fix/stuckdoors
```

4.6.314 fix/tile-occupancy

Tags: fort | bugfix | map

Command: `fix/tile-occupancy`

Fix tile occupancy flags.

This tool clears bad occupancy flags at the selected tile. It is useful for getting rid of phantom "building present" messages when trying to build something in an obviously empty tile.

To use, select a tile with the in-game "look" (`k` mode) cursor.

Usage

```
fix/tile-occupancy
```

4.6.315 gui/advfort

Tags: adventure | gameplay

Command: `gui/advfort`

Perform fort-like jobs in adventure mode.

Keybinding: CtrlT in `dungeonmode`

This script allows performing jobs in adventure mode. For interactive help, press `?` while the script is running.

Warning: Note that changes are only saved for non-procedural sites, i.e. caves, camps, and player forts. Other sites will lose the changes you make when you leave the area.

Usage

```
gui/advfort [<job type>] [<options>]
```

You can specify a job type (e.g. `Dig`, `FellTree`, etc.) to pre-select it in the `gui/advfort` UI. Otherwise you can just select the desired job type in the UI after it comes up.

Examples

`gui/advfort` Brings up the GUI for interactive job selection. Items that dwarves can use will be available in item selection lists.

`gui/advfort -e` Brings up the GUI for interactive job selection. Items that the adventurer's civilization can use will be available in item selection lists.

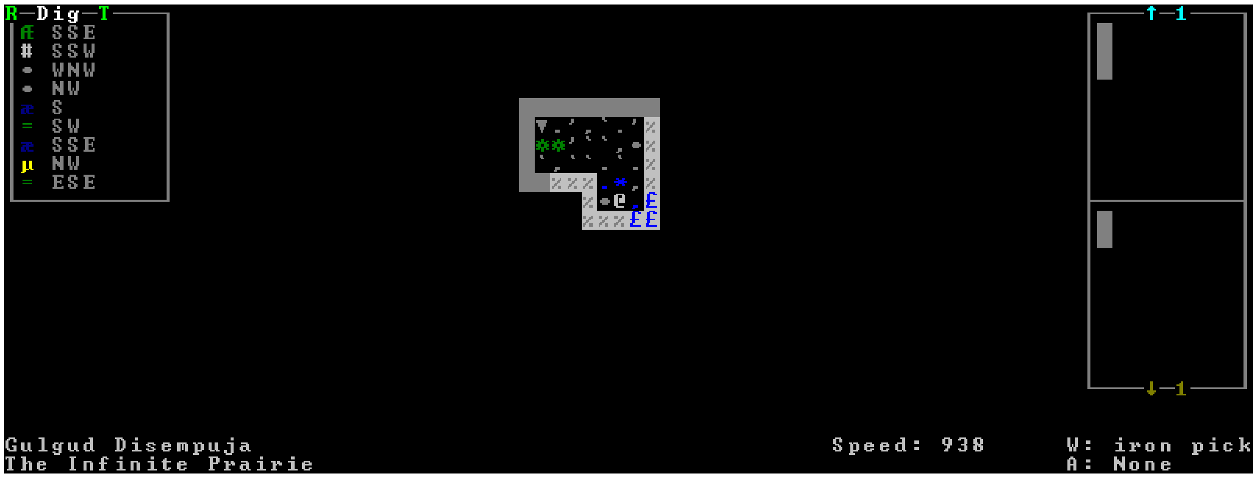
Options

`-c`, `--cheat` Relaxes item requirements for buildings (e.g. you can make walls from bones, which you cannot normally do).

`-e [NAME]`, `--entity [NAME]` Use the given civ (specified as an entity raw ID) to determine which resources are usable. Defaults to `MOUNTAIN` (i.e. Dwarf). If `-e` is specified but the entity name is omitted then it defaults to the adventurer's civ.

Screenshot

Here is an example of a player digging in adventure mode:



4.6.316 gui/autobutcher

Tags: fort | auto | fps | animals

Command: `gui/autobutcher`

Automatically butcher excess livestock.

Keybinding: ShiftB in `pet/List/Unit`

This is an in-game interface for *autobutcher*, which allows you to set thresholds for how many animals you want to keep and will automatically butcher the extras.

Usage

```
gui/autobutcher
```

4.6.317 gui/autogems

Tags: fort | auto | workorders

Command: `gui/autogems`

Automatically cut rough gems.

This is a frontend for the *autogems* plugin that allows interactively configuring the gem types that you want 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.

Usage

```
gui/autogems
```

4.6.318 gui/blueprint

Tags: fort | design | buildings | map | stockpiles

Command: `gui/blueprint`

Record a live game map in a quickfort 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, but, if specified, they will override the initial values set in the interface. See the *blueprint* documentation for information on the possible options.

Examples

`gui/blueprint` Start the blueprint GUI with default initial values.

`gui/blueprint tavern dig build --format pretty` Start the blueprint GUI with the blueprint name pre-set to `tavern`, the `dig` and `build` blueprint phases enabled (and all other phases turned off), and with the output format set to `pretty`.

4.6.319 gui/choose-weapons

Tags: fort | productivity | military

Command: `gui/choose-weapons`

Ensure military dwarves choose appropriate weapons.

Keybinding: CtrlW in `layer_military/Equip/Customize/View`

Activate in the *Equip->View/Customize* page of the military screen.

A weapon specification of "individual choice" is unreliable when there is a weapon shortage. Your military dwarves often end up equipping weapons with which they have no experience using.

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

Usage

```
gui/choose-weapons
```

4.6.320 gui/clone-uniform

Tags: fort | productivity | military

Command: `gui/clone-uniform`

Duplicate an existing military uniform.

Keybinding: CtrlC in `layer_military/Uniforms`

When invoked, this tool 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.

Usage

```
gui/clone-uniform
```

4.6.321 gui/color-schemes

Tags: graphics

Command: `gui/color-schemes`

Modify the colors in the DF UI.

This is an in-game interface for *color-schemes*, which allows you to modify the colors in the Dwarf Fortress interface. This script must be called from either the title screen (shown when you first start the Dwarf Fortress game) or a fortress main map screen.

Usage

```
gui/color-schemes
```

4.6.322 gui/companion-order

Tags: adventure | interface

Command: `gui/companion-order`

Issue orders to companions.

Keybinding: ShiftO in `dungeonmode`

This tool allows you to issue orders to your adventurer's companions. Select which companions to issue orders to with lower case letters (green when selected), then issue orders with upper case letters. You must be in look or talk mode to issue commands that refer to a tile location (e.g. move/equip/pick-up).

Usage

```
gui/companion-order [-c]
```

Call with `-c` to enable "cheating" orders (see below).

Orders

move Order selected companions to move to a location. If the companions are currently following you, they will move no more than 3 tiles from you.

equip Try to equip the items on the ground at the selected tile.

pick-up Try to take items at the selected tile and wield them.

unequip Remove and drop equipment.

unwield Drop held items.

wait Temporarily leave party.

follow Rejoin the party after "wait".

leave Permanently leave party (can be rejoined by talking).

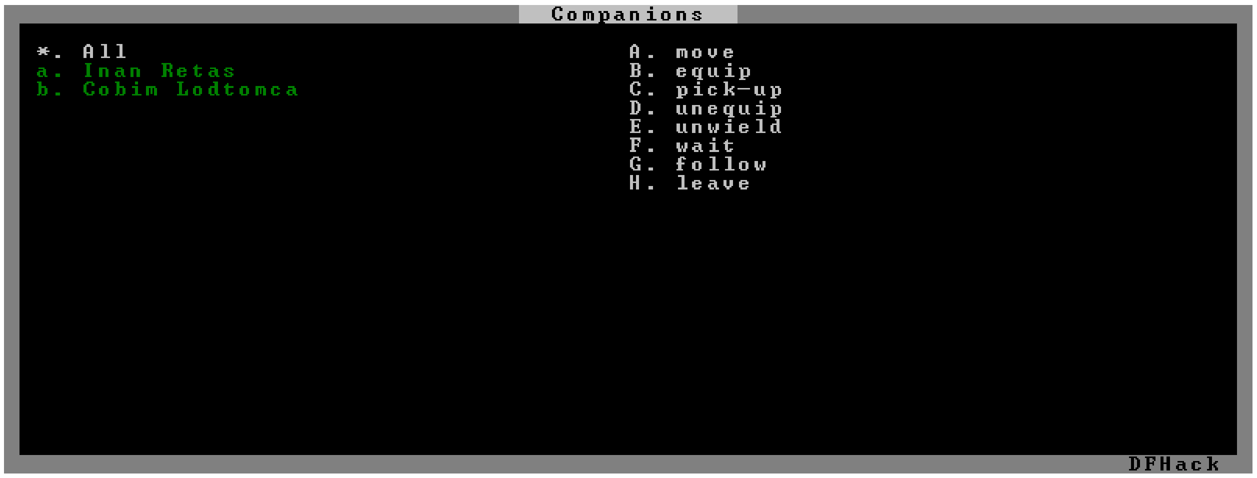
If `gui/companion-order` was called with the `-c` option, the following orders will be available:

patch up Heal all wounds.

get in Ride thing (e.g. minecart) at cursor. There may be some graphical anomalies when pushing a minecart with a companion riding in it.

Screenshot

Here is a screenshot of the tool in action:



```
                              Companions
*. All                                  A. move
a. Inan Retas                           B. equip
b. Cobim Lodtomca                       C. pick-up
                                        D. unequip
                                        E. unwield
                                        F. wait
                                        G. follow
                                        H. leave
                                                                DFHack
```

4.6.323 gui/confirm-opts

Tags: fort | productivity | interface

Command: `gui/confirm-opts`

Configure which confirmation dialogs are enabled.

This tool is a basic configuration interface for the *confirm* plugin. You can see current state, and you can interactively choose which confirmation dialogs are enabled.

Usage

```
gui/confirm-opts
```

4.6.324 gui/cp437-table

Tags: dfhack

Command: `gui/cp437-table`

Virtual keyboard for typing with the mouse.

Keybinding: CtrlShiftK

This tool provides an in-game virtual keyboard. You can choose from all the characters that DF supports (code page 437). Just click on the characters to build the text that you want to send to the parent screen. The text is sent as soon as you hit `Enter`, so make sure there is a text field selected before starting this UI!

Usage

```
gui/cp437-table
```

4.6.325 gui/create-item

Tags: fort | armok | items

Command: `gui/create-item`

Magically summon any item.

This tool provides a graphical interface for creating items of your choice. It walks you through the creation process with a series of prompts, asking you for the type of item, the material, the quality, and (if `--multi` is passed on the commandline) the quantity.

Be sure to select a unit before running this tool so the created item can have a valid “creator” assigned.

See also *createitem* or *modtools/create-item* for different interfaces for creating items.

Usage

```
gui/create-item [<options>]
```

Examples

gui/create-item --multi Only provide options for creating items that normally exist in the game. Also include the prompt for quantity so you can create more than just one item at a time.

gui/create-item --unrestricted Create one item made of anything in the game. For example, you can create a bar of vomit, if you please.

Options

--multi Also prompt for the quantity of items to create.

--unit <id> Use the specified unit as the "creator" of the generated item instead of the selected unit.

--unrestricted Don't restrict the material options to only those that are normally appropriate for the selected item type.

--startup Instead of showing the item creation interface, start monitoring reactions for a modded reaction with a code of `DFHACK_WISH`. When a reaction with that code is completed, show the item creation gui. This allows you to mod in "wands of wishing" that can let your adventurer make wishes for items.

4.6.326 gui/create-tree

Tags: fort | armok | plants

Command: `gui/create-tree`

Create a tree.

This tool provides a graphical interface for creating trees.

Place the cursor wherever you want the tree to appear before you run the script.

Then, select the desired tree type from the list, and then the age. If you just hit `Enter` for the age, it will default to 1 (the youngest age for a non-sapling tree).

For full-grown trees, try an age of 100.

Usage

```
gui/create-tree
```

4.6.327 gui/dfstatus

Tags: fort | inspection

Command: `gui/dfstatus`

Show a quick overview of critical stock quantities.

Keybinding: CtrlShiftI in `dwarfmode/Default|dfhack/lua/dfstatus`

This tool show a quick overview of stock quantities for:

- drinks
- wood
- fuel
- prepared_meals
- tanned_hides
- cloth
- various metal bars

Sections can be enabled/disabled/configured by modifying `dfhack-config/dfstatus.lua`. In particular, you can customize the list of metals that you want to report on.

Usage

```
gui/dfstatus
```

4.6.328 gui/extended-status

Tags: fort | inspection | interface

Command: `gui/extended-status`

Add information on beds and bedrooms to the status screen.

Adds an additional page to the `z` status screen where you can see information about beds, bedrooms, and whether your dwarves have bedrooms of their own.

Usage

`enable gui/extended-status` Show a hotkey on the status screen to invoke the bedroom_list screen.

`gui/extended-status bedroom_list` Show the bedroom status screen.

4.6.329 gui/family-affairs

Tags: fort | armok | inspection | units

Command: `gui/family-affairs`

Inspect or meddle with romantic relationships.

This tool provides 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.

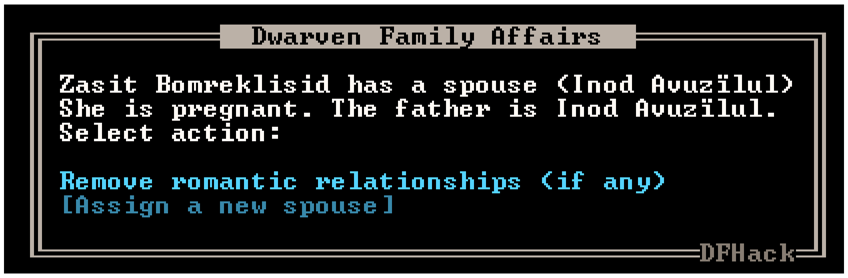
Usage

`gui/family-affairs [unitID]` Show GUI for the selected unit, or the unit with the specified unit ID.

`gui/family-affairs divorce [unitID]` Remove all spouse and lover information from the unit and their partner.

`gui/family-affairs [unitID] [unitID]` Divorce the two specified units and their partners, then arrange for the two units to marry.

Screenshot



```
Dwarven Family Affairs

Zasit Bomreklisid has a spouse (Inod Avuzïlul)
She is pregnant. The father is Inod Avuzïlul.
Select action:

Remove romantic relationships (if any)
[Assign a new spouse]
                                        DFHack
```

4.6.330 gui/gm-editor

Tags: dfhack | armok | inspection | animals | buildings | items | jobs | map | plants | stockpiles | units | workorders

Command: `gui/gm-editor`

Inspect and edit DF game data.

This editor allows you to inspect or modify almost anything in DF. Press ? for in-game help.

Usage

`gui/gm-editor` Open the editor on whatever is selected or viewed (e.g. unit/item description screen)

`gui/gm-editor <lua expression>` Evaluate a lua expression and opens the editor on its results.

`gui/gm-editor dialog` Show an in-game dialog to input the lua expression to evaluate. Works the same as the version above.

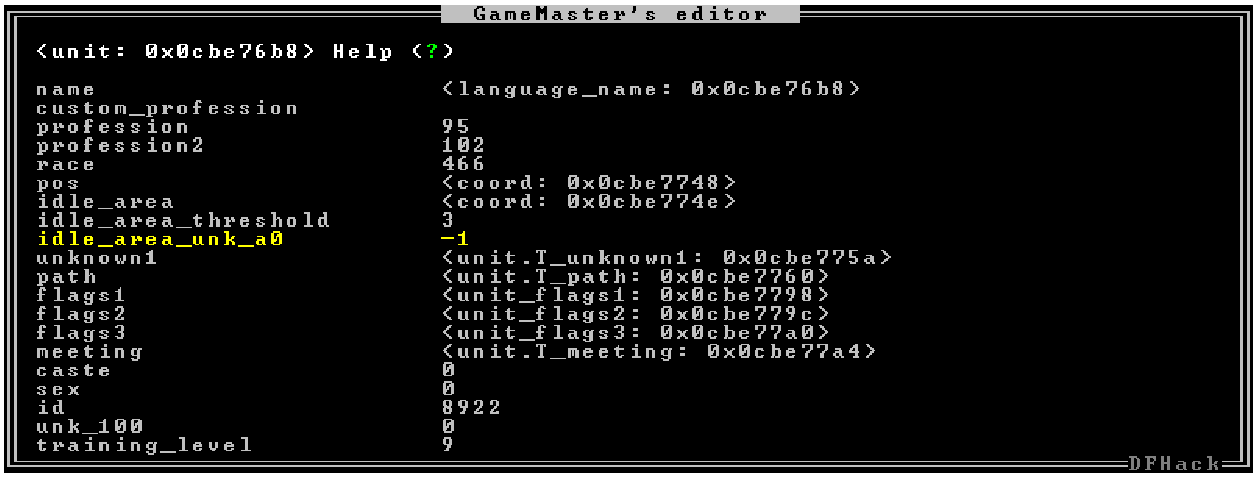
`gui/gm-editor toggle` Hide (if shown) or show (if hidden) the editor at the same position you left it.

Examples

`gui/gm-editor` Opens the editor on the selected unit/item/job/workorder/etc.

`gui/gm-editor df.global.world.items.all` Opens the editor on the items list.

Screenshot



```
GameMaster's editor
<unit: 0x0cbe76b8> Help (?)

name                        <language_name: 0x0cbe76b8>
custom_profession
profession                  95
profession2                 102
race                        466
pos                         <coord: 0x0cbe7748>
idle_area                   <coord: 0x0cbe774e>
idle_area_threshold         3
idle_area_unk_a0            -1
unknown1                    <unit.T_unknown1: 0x0cbe775a>
path                        <unit.T_path: 0x0cbe7760>
flags1                      <unit_flags1: 0x0cbe7798>
flags2                      <unit_flags2: 0x0cbe779c>
flags3                      <unit_flags3: 0x0cbe77a0>
meeting                     <unit.T_meeting: 0x0cbe77a4>
caste                       0
sex                         0
id                          8922
unk_100                     0
training_level              9
                                                                DFHack
```

4.6.331 gui/gm-unit

Tags: dfhack | armok | inspection | animals | units

Command: `gui/gm-unit`

Inspect and edit unit attributes.

This tool provides an editor for unit attributes. Select a unit in-game before running this command or pass a target unit ID on the commandline.

Usage

```
gui/gm-unit [unit id]
```

4.6.332 gui/guide-path

Tags: fort | inspection | map

Command: `gui/guide-path`

Visualize minecart guide paths.

Keybinding: AltP in `dwarfmode/Hauling/DefineStop/Cond/Guide`

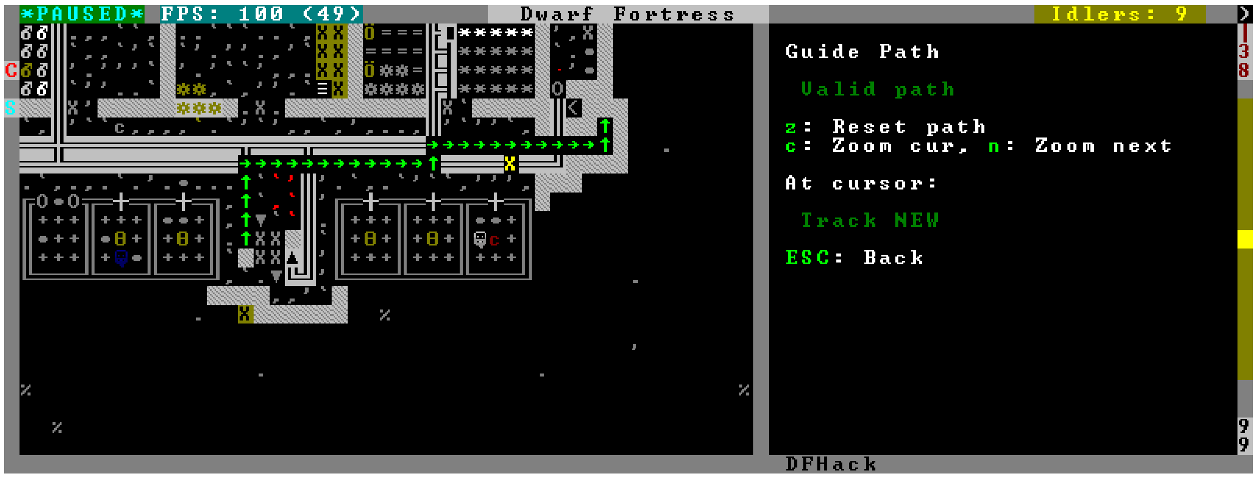
This tool displays the cached path that will be used by the minecart guide order. The game computes this path when the order is executed for the first time, so you have to wait for a dwarf to have started pushing the minecart at least once before you can visualize the path.

To use, select a *Guide* order in the *Hauling* menu with the cursor and run this tool.

Usage

```
gui/guide-path
```

Screenshot



```
Guide Path

 Valid path

z: Reset path
c: Zoom cur, n: Zoom next

At cursor:

 Track NEW

ESC: Back
```

4.6.333 gui/kitchen-info

Tags: fort | inspection

Command: `gui/kitchen-info`

Show food item uses in the kitchen status screen.

This tool is an overlay that adds more info to the Kitchen screen, such as the potential alternate uses of the items that you could mark for cooking.

Usage

```
enable gui/kitchen-info
```

4.6.334 gui/launcher

Tags: dfhack

Command: `gui/launcher`

In-game DFHack command launcher with integrated help.

Keybinding: `

Keybinding: CtrlShiftD

This tool is the primary GUI interface for running DFHack commands. You can open it from any screen with the ` hotkey. Tap ` again (or hit `Esc`) to close. Users with keyboard layouts that make the ` key difficult (or impossible) to press can use the alternate hotkey of `CtrlShiftD`.

Usage

```
gui/launcher [initial commandline]
```

Examples

`gui/launcher` Open the launcher dialog with a blank initial commandline.

`gui/launcher prospect --show ores,veins` Open the launcher dialog with the edit area pre-populated with the given command, ready for modification or running. Tools related to `prospect` will appear in the autocomplete list, and help text for `prospect` will be displayed in the help area.

Editing and running commands

Enter the command you want to run by typing its name. If you want to start over, `CtrlC` will clear the line. When you are happy with the command, hit `Enter` or click on the `run` button to run it. Any output from the command will appear in the help area. If you want to run the command but close the dialog immediately so you can get back to the game, use `ShiftEnter` or hold down the `Shift` key and click on the `run` button instead. The dialog also closes automatically if you run a command that brings up a new GUI screen. In any case, the command output will also be written to the DFHack terminal console if you need to find it later.

If your keyboard layout makes any key impossible to type (such as `[` and `]` on German QWERTZ keyboards), use `CtrlShiftK` to bring up the on-screen keyboard. You can "type" the characters you need by clicking on the character with the mouse.

Autocomplete

As you type, autocomplete options for DFHack commands appear in the right column. If the first word of what you've typed matches a valid command, then the autocomplete options will also include commands that have similar functionality to the one that you've named. Click on an autocomplete list option to select it or cycle through them with `Tab` and `ShiftTab`.

Context-sensitive help

When you start `gui/launcher` without parameters, it shows some useful information in the help area about how to get started with browsing DFHack tools by their category *tags*.

Once you have typed (or autocompleted) a word that matches a valid command, the help area shows the help for that command, including usage instructions and examples. You can scroll the help text by half-pages by clicking on the scrollbar or with `PgUp` and `PgDn`. You can also scroll line by line with `CtrlUp` and `CtrlDown`, or by clicking on the scrollbar arrows.

Command history

`gui/launcher` keeps a history of commands you have run to let you quickly run those commands again. You can scroll through your command history with the `Up` and `Down` cursor keys, or you can search your history for something specific with the `AltS` hotkey. When you hit `AltS`, start typing to search your history for a match. To find the next match for what you've already typed, hit `AltS` again. You can run the matched command immediately with `Enter` (or `ShiftEnter`), or hit `Esc` to edit the command before running it.

Dev mode

By default, commands intended for developers and modders are filtered out of the autocomplete list. You can toggle this filtering by hitting `CtrlD` at any time.

Autocomplete tab speed

By default in DF, holding down the `Tab` key is the same as just pressing it once. This makes scanning quickly through the autocomplete list with the keyboard quite slow. To speed things up, you can go to the DF keybinding configuration screen (hit `Esc` from the main map and select `Key Bindings` from the menu), select `General`, and then `Change tab or highlight selection`. If you change the configuration for that item to `Immediate repeat`, then holding `Tab` will allow you to scan through the autocomplete list more quickly. Be sure to also change the configuration for the next item, `Change tab or highlight selection, secondary`, to also allow `ShiftTab` to work the same way.

Back at the main Key Bindings screen, select `Save and exit` to get back to the game.

4.6.335 gui/liquids

Tags: fort | armok | map

Command: `gui/liquids`

Interactively paint liquids or obsidian onto the map.

Keybinding: AltL in `dwarfmode/LookAround`

This tool 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**. Be sure the settings are what you want before hitting Enter!

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 cycles through modes for adding water, magma, obsidian walls/floors, or modifying liquid tile properties.

When painting liquids, you can select the desired level with `+-`, and you can choose among setting it exactly, only increasing, or only decreasing with `s`.

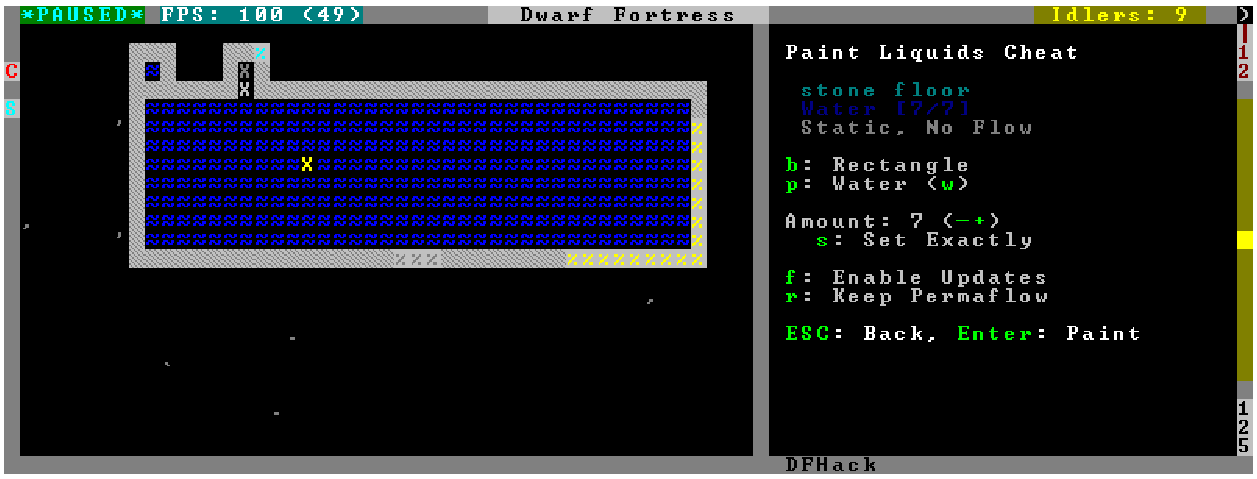
In addition, `f` allows disabling or enabling the flowing water computations for an area, and `r` operates on the "permanent flow" property that makes rivers power water wheels even when full and technically not flowing.

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

Usage

```
gui/liquids
```

Screenshot



```
*PAUSED* FPS: 100 (49)        Dwarf Fortress        Idlers: 9

Paint Liquids Cheat

 stone floor
 Water [7/7]
 Static, No Flow

b: Rectangle
p: Water (w)

Amount: 7 (-+)
  s: Set Exactly

f: Enable Updates
r: Keep Permaflow

ESC: Back, Enter: Paint

DFHack
```

4.6.336 gui/load-screen

Tags: dfhack

Command: `gui/load-screen`

Replace DF's continue game screen with a searchable list.

If you tend to have many ongoing games, this tool can make it much easier to load the one you're looking for. It replaces DF's "continue game" screen with a dialog that has search and filter options.

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`).

Also see the `title-start-rename` *tweak* to rename folders in the "start playing" menu.

Usage

```
enable gui/load-screen
```

4.6.337 gui/manager-quantity

Tags: fort | workorders

Command: `gui/manager-quantity`

Set the quantity of the selected manager workorder.

Keybinding: AltQ in `jobmanagement/Main`

There is no way in the base DF game to change the quantity for an existing manager workorder. Select a workorder on the j-m or u-m screens and run this tool to set a new total requested quantity. The number of items remaining in the workorder will be modified accordingly.

Usage

```
gui/manager-quantity
```

4.6.338 gui/mass-remove

Tags: fort | productivity | buildings | stockpiles

Command: `gui/mass-remove`

Mass select buildings and constructions to suspend or remove.

This tool lets you remove buildings/constructions or suspend/unsuspend construction jobs using a mouse-driven box selection.

The following marking modes are available.

Suspend Suspend the construction of a planned building/construction.

Unsuspend Resume the construction of a suspended planned building/construction.

Remove Construction Designate a construction (wall, floor, etc.) for removal. This is similar to the native Designate->Remove Construction menu in DF.

Unremove Construction Cancel removal of a construction (wall, floor, etc.).

Remove Building Designate a building (door, workshop, etc) for removal. This is similar to the native Set Building Tasks/Prefs->Remove Building menu in DF.

Unremove Building Cancel removal of a building (door, workshop, etc.).

Remove All Designate both constructions and buildings for removal, and deletes planned buildings/constructions.

Unremove All Cancel removal designations for both constructions and buildings.

Usage

```
gui/mass-remove
```

4.6.339 gui/mechanisms

Tags: fort | inspection | buildings

Command: `gui/mechanisms`

List mechanisms and links connected to a building.

Keybinding: CtrlM in `dwarfmode/QueryBuilding/Some`

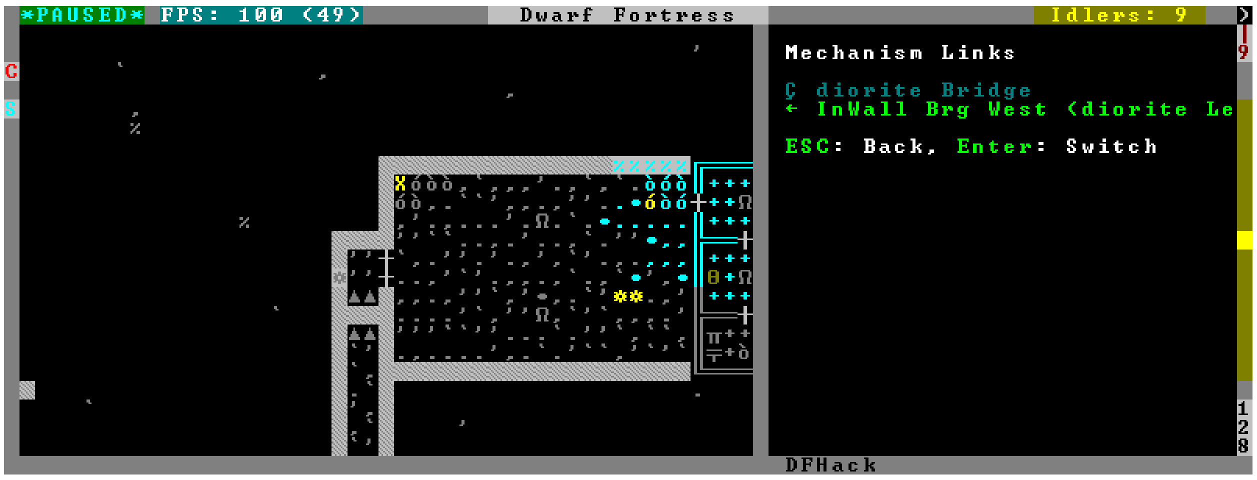
This convenient tool lists the mechanisms connected to the building and the buildings linked via the mechanisms. Navigating the list centers the view on the relevant linked building.

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.

Usage

```
gui/mechanisms
```

Screenshot



4.6.340 gui/mod-manager

Tags: dfhack

Command: `gui/mod-manager`

Easily install and uninstall mods.

This tool provides a simple way to install and remove small mods that you have downloaded from the internet – or have created yourself! Several mods are available here. Mods that you want to manage with this tool should go in the `mods` subfolder under your main DF folder.

The mod manager must be invoked on the Dwarf Fortress title screen, *before* a world is generated. Any mods that you install will only affect worlds generated after you install them.

Usage

```
gui/mod-manager
```

Mod format

Each mod must include a lua script that defines the following variables:

name The name that should be displayed in the mod manager list.

author The mod author.

description A description of the mod

Of course, this doesn't actually make a mod - so one or more of the following variables 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 use to patch the `entity_default.txt` file.

patch_init A chunk of text to add to the `init.lua` file in the raws.

patch_dofile A list (table) of files to run from `init.lua`.

patch_files A table of files to patch, each element containing the following subfields:

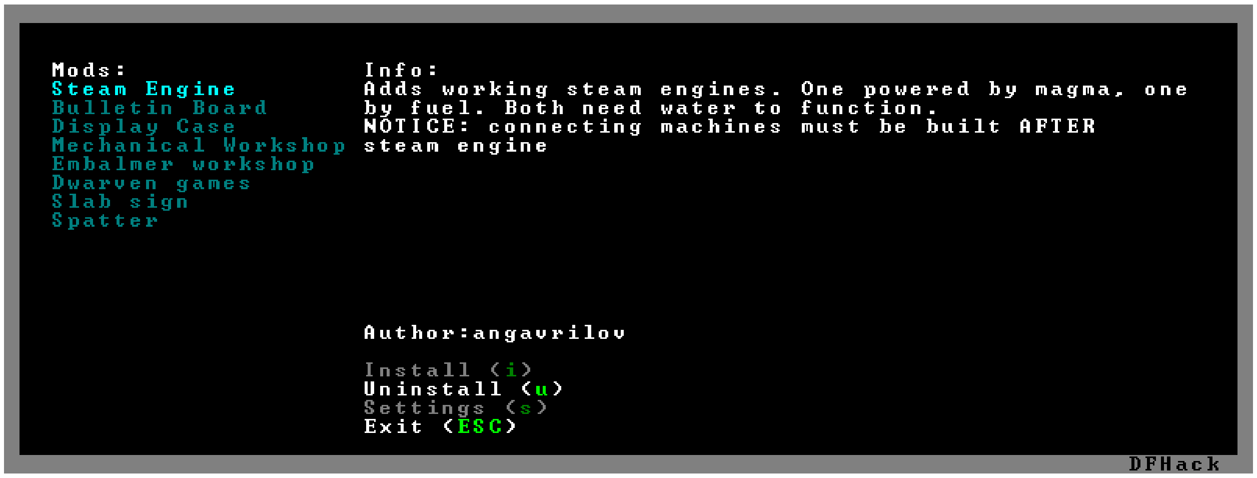
- **filename** A filename (relative to the raws folder) to patch.
- **patch** The text to add.
- **after** A string after which to insert the text.

guard A token that is used in raw files to find additions from this mod and remove them on uninstall.

guard_init A token that is used in the `init.lua` file to find additions from this mod and remove them on uninstall.

[pre|post]_(un)install Callback functions for each installation/uninstallation stage that can be used to trigger more complex install behavior.

Screenshot



```
Mods:                   Info:
Steam Engine            Adds working steam engines. One powered by magma, one
Bulletin Board          by fuel. Both need water to function.
Display Case            NOTICE: connecting machines must be built AFTER
Mechanical Workshop     steam engine
Embalmer workshop
Dwarven games
Slab sign
Spatter



                        Author:angavrilov

                        Install (i)
                        Uninstall (u)
                        Settings (s)
                        Exit (ESC)
                                                                  DFHack
```

4.6.341 gui/overlay

Tags: dfhack | interface

Command: `gui/overlay`

Manage DFHack overlays and overlay widgets.

This is the configuration interface for the *overlay* framework. You can see which overlays are available and which ones are enabled, either globally or just for overlays and widgets associated with the current Dwarf Fortress screen. Each overlay widget will be framed on the screen to identify which widget is which. You can click on the frame highlight and drag the widget around the screen to reposition it, or hit the indicated hotkey and reposition with the cursor keys.

The frame around the currently selected widget will show a highlight to indicate which screen edge the widget is anchored to. For example, if the bottom and right edges of the frame are highlighted, the widget will move relative to the bottom and right screen edge when the DF window is resized.

Usage

```
gui/overlay
```

4.6.342 gui/pathable

Tags: fort | inspection | map

Command: `gui/pathable`

Highlights reachable tiles.

Keybinding: AltShiftP in `dwarfmode/LookAround`

This tool highlights each visible map tile to indicate whether it is possible to path to that tile from the tile at the cursor. You can move the cursor around and the highlight will change dynamically. The highlight is green if pathing is possible and red if not, similar to the highlight DF uses to indicate which tiles can reach the trade depot.

While the UI is active, you can use the following hotkeys to change the behavior:

- **`l`: Lock cursor: when enabled, the movement keys move 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. This** allows you to see the map in its regular shading, if desired.
- **u: Skip unrevealed: when enabled, unrevealed tiles will not be** highlighted 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.

Usage

```
gui/pathable
```

4.6.343 gui/petitions

Tags: fort | inspection

Command: `gui/petitions`

Show information about your fort's petitions.

Show your fort's petitions, both pending and fulfilled.

Usage

```
gui/petitions
```

4.6.344 gui/power-meter

Tags: fort | gameplay | buildings

Command: `gui/power-meter`

Allow pressure plates to measure power.

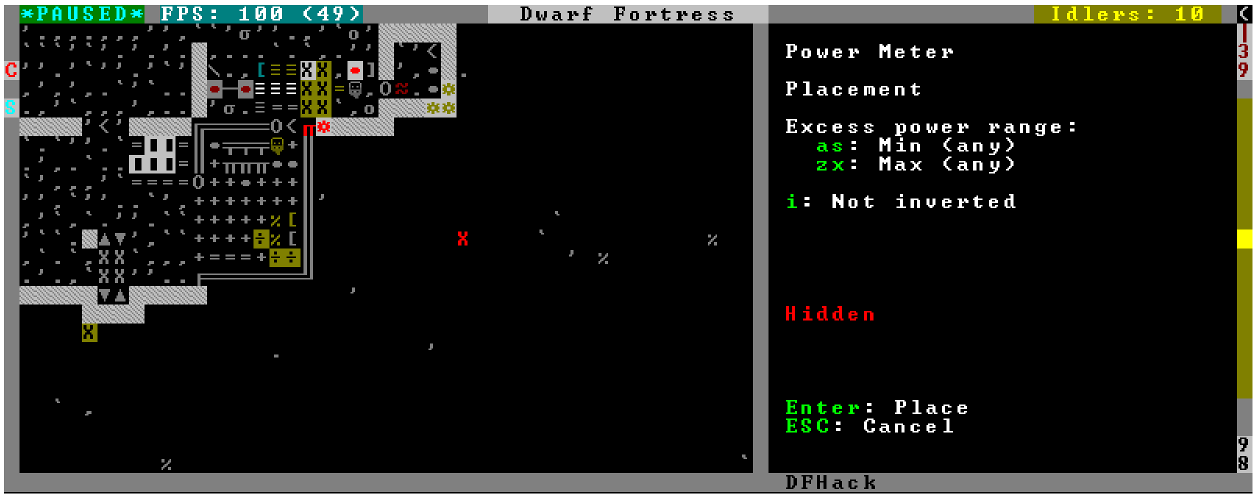
Keybinding: CtrlShiftM in `dwarfmode/Build/Position/Trap`

If you run this tool after selecting *Pressure Plate* in the build menu, you will build a power meter building instead of a regular pressure plate. This means that, once the building is built, you can select it to see a readout of how much power is being supplied to gear boxes built in the four adjacent N/S/W/E tiles.

Usage

```
gui/power-meter
```

Screenshot



```
Power Meter

Placement

Excess power range:
   as: Min (any)
   zx: Max (any)

i: Not inverted



Hidden



Enter: Place
ESC: Cancel
```

4.6.345 gui/prerelease-warning

Tags: dfhack

Command: `gui/prerelease-warning`

Shows a warning if you are using a pre-release build of DFHack.

This tool shows a warning on world load for pre-release builds.

Usage

```
gui/prerelease-warning [force]
```

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.

4.6.346 gui/quantum

Tags: fort | productivity | stockpiles

Command: `gui/quantum`

Quickly and easily create quantum stockpiles.

This tool provides a visual, interactive interface for creating quantum stockpiles.

Quantum stockpiles simplify fort management by allowing a small stockpile to contain an infinite number of items. This reduces the complexity of your storage design, lets your dwarves be more efficient, and increases FPS.

Quantum stockpiles work by linking a "feeder" stockpile to a one-tile minecart hauling route. As soon as an item from the feeder stockpile is placed in the minecart, the minecart is tipped and all items land on an adjacent tile. The single-tile stockpile in that adjacent tile that holds all the items is your quantum stockpile.

Before you run this tool, create and configure your "feeder" stockpile. The size of the stockpile determines how many dwarves can be tasked with bringing items to this quantum stockpile. Somewhere between 1x3 and 5x5 is usually a good size. Make sure to assign an appropriate number of wheelbarrows to feeder stockpiles that will contain heavy items like corpses, furniture, or boulders.

The UI will walk you through the steps:

1) Select the feeder stockpile by clicking on it or selecting it with the cursor and hitting Enter.
2) Configure the orientation of your quantum stockpile and select whether to allow refuse and corpses with the onscreen options.
3) Select a spot on the map to build the quantum stockpile by clicking on it or moving the cursor so the preview "shadow" is in the desired location. Then hit `Enter`.

If there are any minecarts available, one will be automatically assigned to the hauling route. If you don't have a free minecart, `gui/quantum` will enqueue a manager order to make one for you. Once it is built, run *assign-minecarts all* to assign it to the route, or enter the (h)auling menu and assign one manually. The quantum stockpile needs a minecart to function.

Quantum stockpiles work much more efficiently if you add the following line to your `dfhack-config/init/onMapLoad.init` file:

```
prioritize -a StoreItemInVehicle
```

This prioritizes moving of items from the feeder stockpile to the minecart. Otherwise, the feeder stockpile can get full and block the quantum pipeline.

See the wiki for more information on quantum stockpiles.

Usage

```
gui/quantum
```

4.6.347 gui/quickcmd

Tags: dfhack

Command: `gui/quickcmd`

Quickly run saved commands.

This tool lets you keep a list of commands that you use often enough to not want to type it every time, but not often enough to be bothered to find a free keybinding. You can edit and save the commands from the UI.

Also see *gui/launcher*, whose history search functionality allows you to run commands from your command history quickly and easily.

Usage

```
gui/quickcmd
```

4.6.348 gui/quickfort

Tags: fort | design | productivity | buildings | map | stockpiles

Command: `gui/quickfort`

Apply pre-designed blueprints to your fort.

Keybinding: CtrlShiftQ in `dwarfmode`

Keybinding: AltF in `dwarfmode`

This is the 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 or the hotkeys to rotate and repeat the blueprint up or down z-levels. 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/aquifer_tap.csv -n /dig`), 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.

To rotate or flip the blueprint around, enable transformations with `t` and use `Ctrl` with the arrow keys to add a transformation step:

`CtrlLeft`: Rotate counterclockwise (ccw) `CtrlRight`: Rotate clockwise (cw) `CtrlUp`: Flip vertically (vflip) `CtrlDown`: Flip horizontally (hflip)

If you have applied several transformations, but there is a shorter sequence that can be used to get the blueprint into the configuration you want, it will automatically be used. For example, if you rotate clockwise three times, `gui/quickfort` will shorten the sequence to a single counterclockwise rotation for you.

Any settings you set in the UI, such as search terms for the blueprint list or transformation options, are saved for the next time you open the UI. This is for convenience when you are applying multiple related blueprints that need to have the same transformation and repetition settings when they are applied.

Examples

`gui/quickfort` Open the quickfort interface with saved settings.

`gui/quickfort dreamfort` Open with a custom filter that shows only blueprints that match the string `dreamfort`.

`gui/quickfort myblueprint.csv` Open with the `myblueprint.csv` blueprint pre-loaded.

4.6.349 gui/rename

Tags: fort | productivity | buildings | stockpiles | units

Command: `gui/rename`

Give buildings and units new names, optionally with special chars.

Keybinding: CtrlShiftN in `dwarfmode|unit|unitlist|joblist|dungeon_monsterstatus|layer_unit_relationship|iten`

Keybinding: CtrlShiftT -> `"gui/rename unit-profession"` in `dwarfmode|unit|unitlist|joblist|dungeon_monsterstatu`

Once you select a target on the game map, this tool allows you to rename it. It is more powerful than the in-game rename functionality since it allows you to use special characters (like diamond symbols), and it also allows you to rename enemies and overwrite animal species strings.

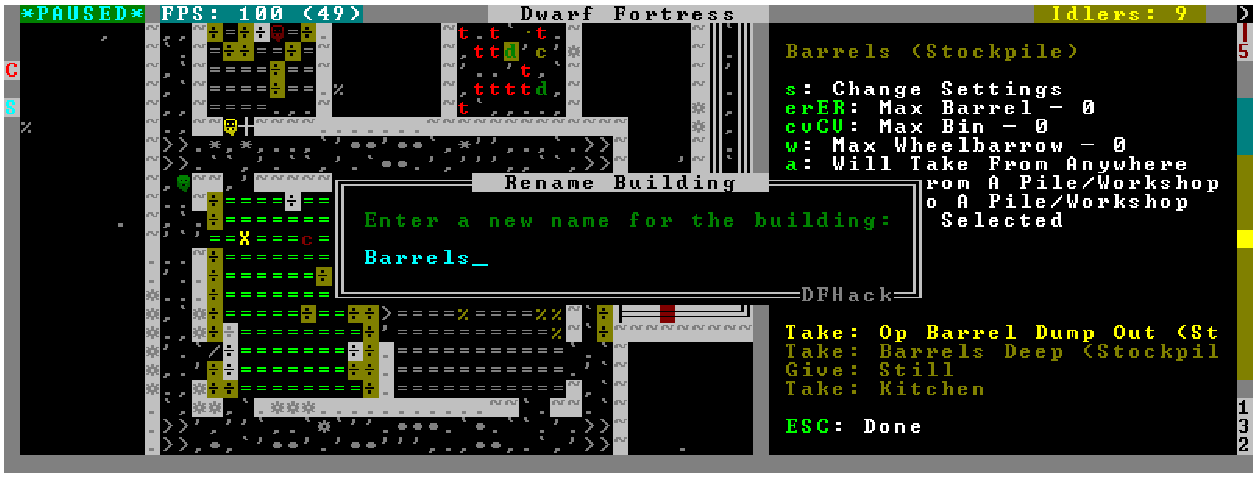
This tool supports renaming units, zones, stockpiles, workshops, furnaces, traps, and siege engines.

Usage

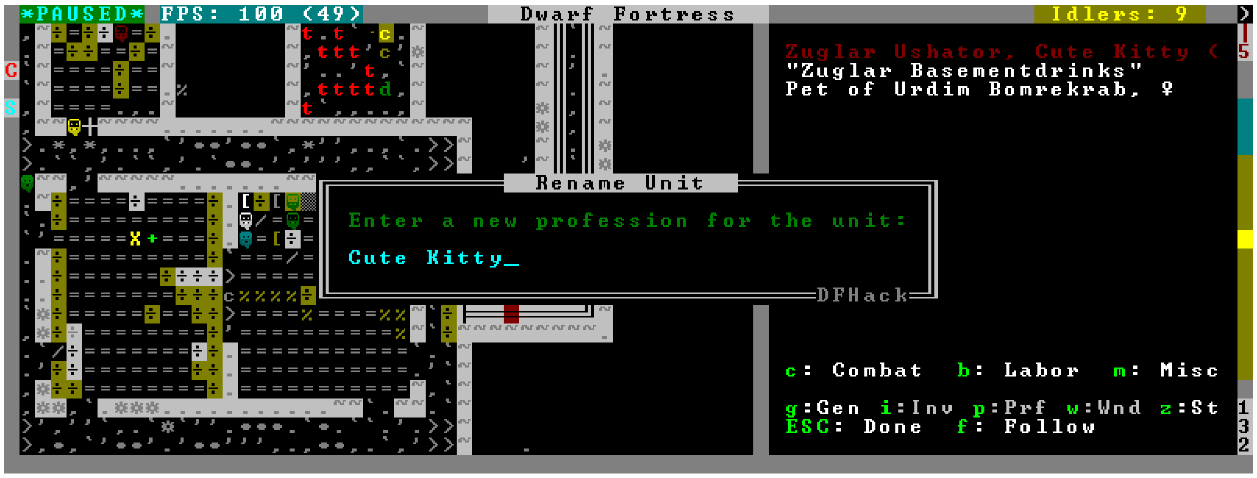
gui/rename Renames the selected building, zone, or unit.

gui/rename unit-profession Set the unit profession or the animal species string.

Screenshots



```
*PAUSED* FPS: 100 (49)          Dwarf Fortress          Idlers: 9

Barrels (Stockpile)

s: Change Settings
erER: Max Barrel - 0
cvCV: Max Bin - 0
w: Max Wheelbarrow - 0
a: Will Take From Anywhere
rom A Pile/Workshop
o A Pile/Workshop
 Selected

Rename Building
Enter a new name for the building:
Barrels_
DFHack

Take: Op Barrel Dump Out (St
Take: Barrels Deep (Stockpil
Give: Still
Take: Kitchen

ESC: Done
```



```
*PAUSED* FPS: 100 (49)          Dwarf Fortress          Idlers: 9

Zuglar Ushator, Cute Kitty (
"Zuglar Basementdrinks"
Pet of Urdim Bomrekrab, ♀

Rename Unit
Enter a new profession for the unit:
Cute Kitty_
DFHack

c: Combat   b: Labor   m: Misc

g:Gen  i:Inv  p:Prf  w:Wnd  z:St
ESC: Done   f: Follow
```

4.6.350 gui/room-list

Tags: fort | inspection

Command: `gui/room-list`

Manage rooms owned by a dwarf.

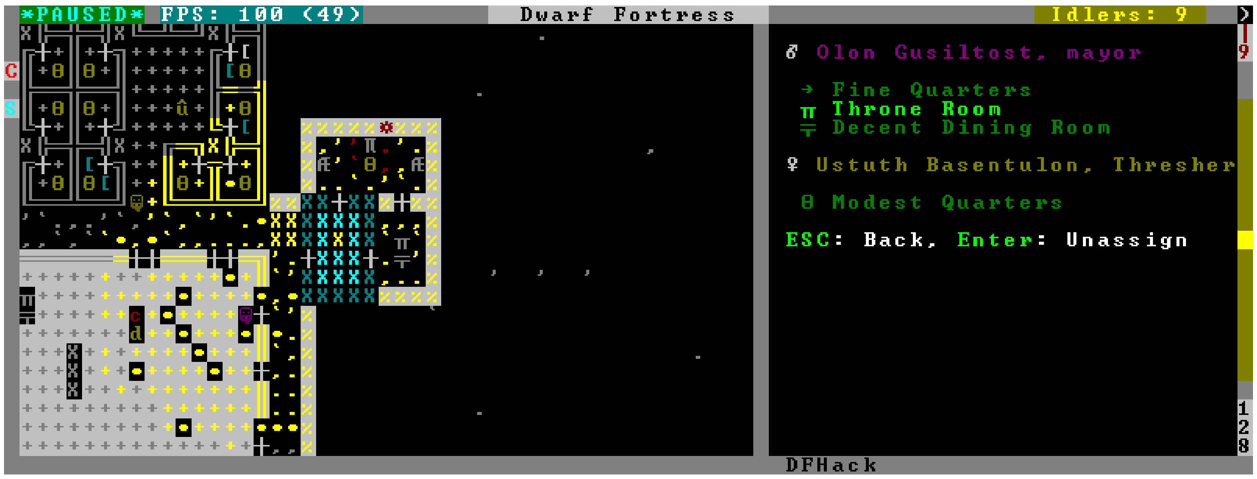
Keybinding: AltR in `dwarfmode/QueryBuilding/Some`

When invoked in q mode with the cursor over an owned room, this tool lists other rooms owned by the same owner, or by the unit selected in the assign list, and allows unassigning them from other rooms.

Usage

```
gui/room-list
```

Screenshot



```
♂ Olon Gusiltost, mayor

 → Fine Quarters
 π Throne Room
 ╤ Decent Dining Room

♀ Ustuth Basentulon, Thresher

 θ Modest Quarters

ESC: Back, Enter: Unassign
```

4.6.351 gui/settings-manager

Tags: dfhack

Command: `gui/settings-manager`

Dynamically adjust global DF settings.

Keybinding: AltS in `title|dwarfmode/Default|dungeonmode`

This tool is an in-game editor for settings defined in `data/init/init.txt` and `data/init/d_init.txt`. Changes are written back to the init files so they will be loaded the next time you start DF. For settings that can be dynamically adjusted, such as the population cap, the active value used by the game is updated immediately.

Editing the population caps will override any modifications made by scripts such as *max-wave*.

Usage

```
gui/settings-manager
```

4.6.352 gui/siege-engine

Tags: fort | gameplay | buildings

Command: `gui/siege-engine`

Extend the functionality and usability of siege engines.

Keybinding: AltA in `dwarfmode/QueryBuilding/Some/SiegeEngine`

This tool is an in-game interface for *siege-engine*, which allows you to link siege engines to stockpiles, restrict operation to certain dwarves, fire a greater variety of ammo, and aim in 3 dimensions.

Run the UI after selecting a siege engine in `q` mode.

The main UI 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 the UI. The area to paste is kept until you quit DF, or until you select another area manually.

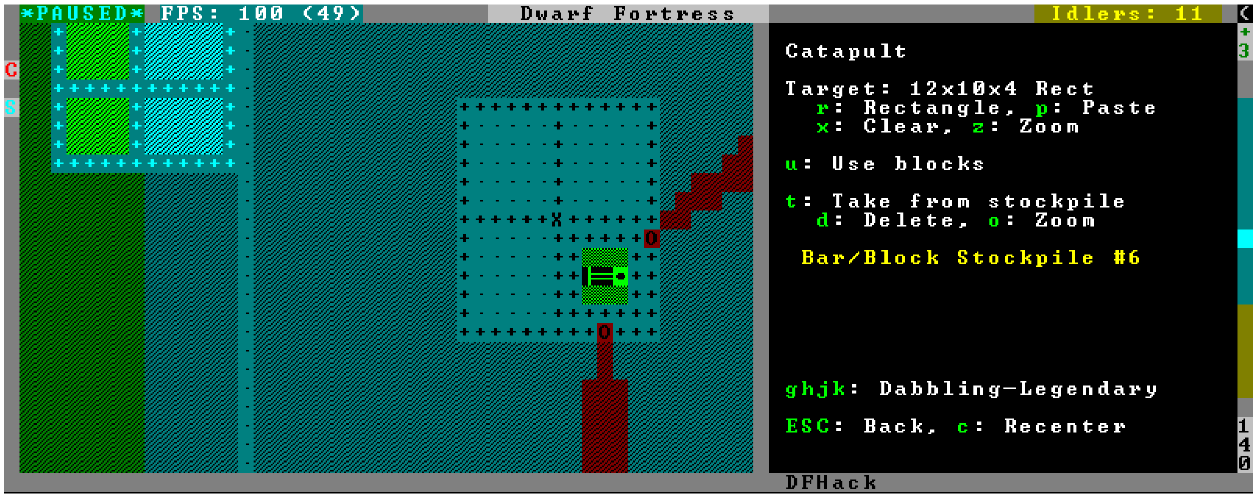
Pressing `t` switches to a mode for selecting a stockpile to take ammo from.

Exiting from the siege engine UI 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 the main map.

Usage

```
gui/siege-engine
```

Screenshot



```
*PAUSED* FPS: 100 (49)        Dwarf Fortress                Idlers: 11

Catapult

Target: 12x10x4 Rect
  r: Rectangle, p: Paste
  x: Clear, z: Zoom

u: Use blocks

t: Take from stockpile
  d: Delete, o: Zoom

 Bar/Block Stockpile #6

ghjk: Dabbling-Legendary

ESC: Back, c: Recenter

DFHack
```

4.6.353 gui/stamper

Tags: fort | design | map

Command: `gui/stamper`

Copy, paste, and transform dig designations.

This tool allows you to copy and paste blocks of dig designations. You can also transform what you have copied by shifting it, reflecting it, rotating it, and/or inverting it. Designations can also be used as brushes to erase other designations and cancel constructions.

Usage

```
gui/stamper
```

4.6.354 gui/stockpiles

Tags: fort | design | stockpiles

Command: `gui/stockpiles`

Import and export stockpile settings.

Keybinding: AltL -> "`gui/stockpiles -load`" in `dwarfmode/QueryBuilding/Some/Stockpile`

Keybinding: AltS -> "`gui/stockpiles -save`" in `dwarfmode/QueryBuilding/Some/Stockpile`

With a stockpile selected in `q` mode, you can use this tool to load stockpile settings from a file or save them to a file for later loading, in this fort or any other.

Saved stockpile configs are stored in the `stocksettings` subfolder in the DF folder.

See *stockpiles* for a commandline interface.

Usage

```
gui/stockpiles
```

4.6.355 gui/teleport

Tags: fort | armok | units

Command: `gui/teleport`

Teleport a unit anywhere.

This tool is a front-end for the *teleport* tool. It allows you to interactively choose a unit to teleport and a destination tile using the in-game cursor.

Usage

```
gui/teleport
```

4.6.356 gui/unit-info-viewer

Tags: fort | inspection | units

Command: `gui/unit-info-viewer`

Display detailed information about a unit.

Keybinding: AltI in `dwarfmode/ViewUnits|unitlist`

Displays information about age, birth, maxage, shearing, milking, grazing, egg laying, body size, and death for the selected unit.

Usage

```
gui/unit-info-viewer
```

4.6.357 gui/workflow

Tags: fort | auto | jobs

Command: `gui/workflow`

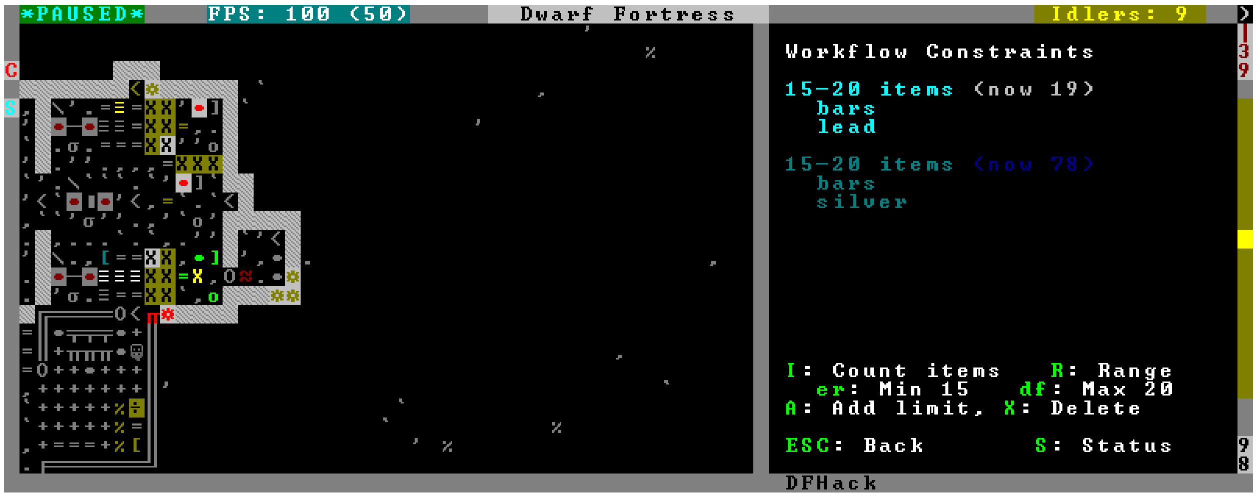
Manage automated item production rules.

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`

This tool provides a simple interface to item production constraints managed by *workflow*. When a workshop job is selected in `q` mode and this tool is invoked, it displays the constraints applicable to the current job and their current status. It also allows you to modify existing constraints or add new ones.

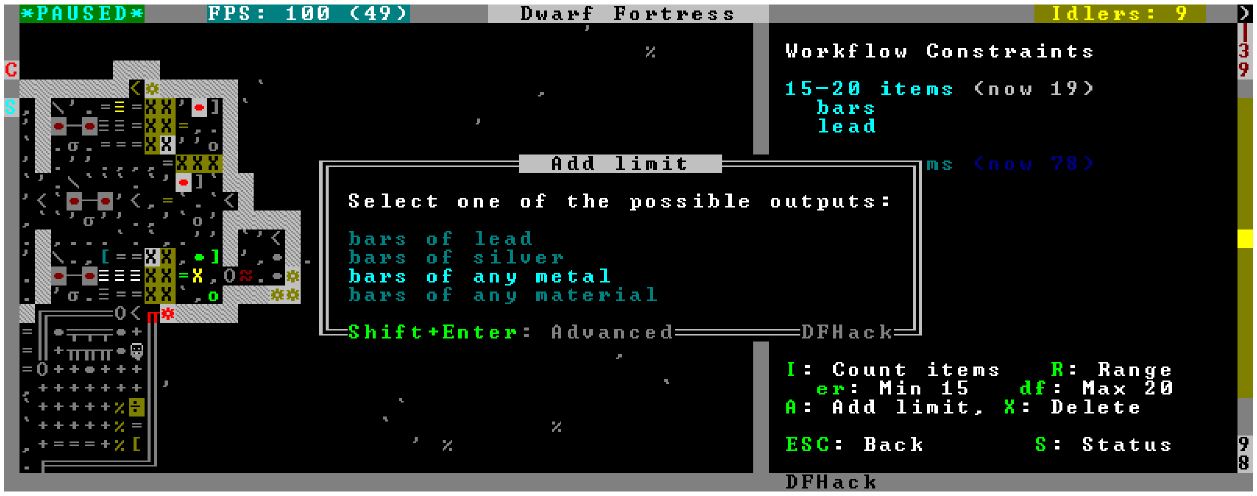


```
*PAUSED*    FPS: 100 (50)    Dwarf Fortress        Idlers: 9

Workflow Constraints

15-20 items (now 19)
  bars
  lead

15-20 items (now 78)
  bars
  silver

I: Count items    R: Range
  er: Min 15    df: Max 20
A: Add limit, X: Delete

ESC: Back         S: Status
DFHack
```

A constraint is a target range to be compared against either the number of individual items or the number of item stacks. It also includes an item type and, optionally, a material. When the current stock 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. If there are multiple constraints, being out of the range of any constraint will cause the job to be suspended.

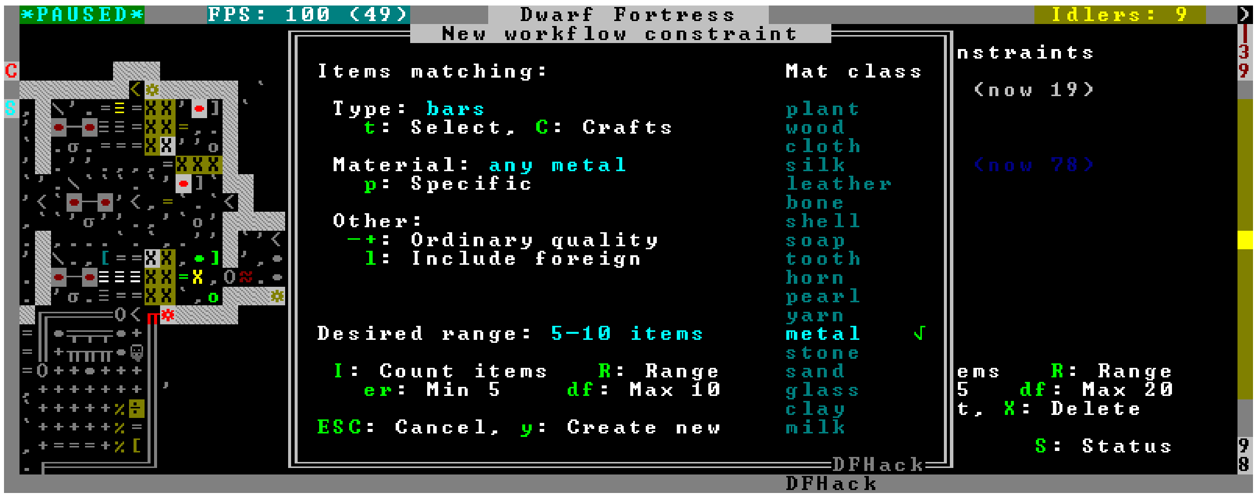
Pressing `I` switches the current constraint between counting stacks and counting individual items. Pressing `R` lets you input the range directly, or `e`, `r`, `d`, `f` incrementally adjusts the bounds.

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.



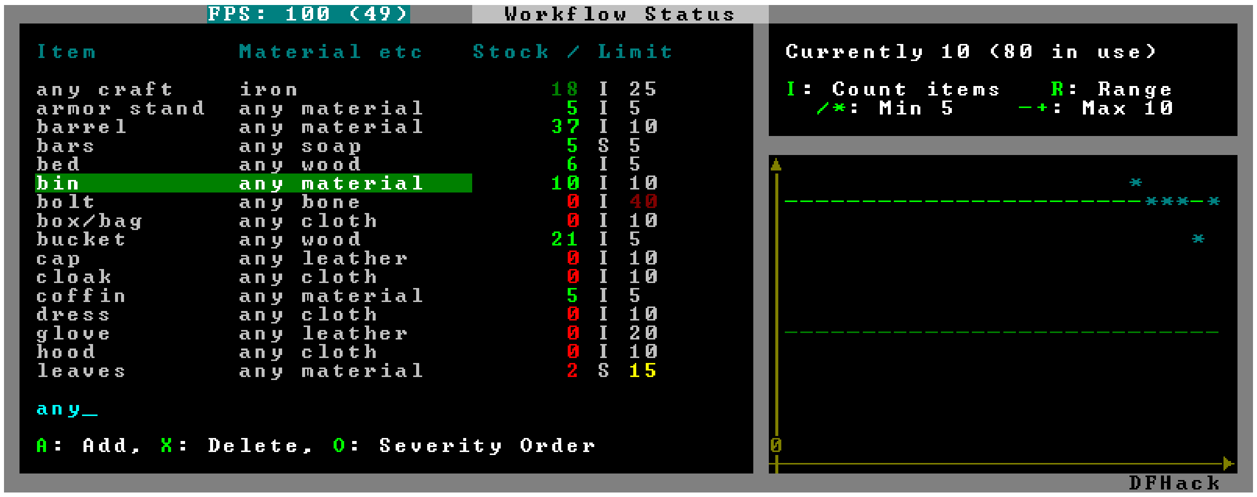
```
*PAUSED*    FPS: 100 (49)    Dwarf Fortress        Idlers: 9

Workflow Constraints

15-20 items (now 19)
  bars
  lead

Add limit

Select one of the possible outputs:

bars of lead
bars of silver
bars of any metal
bars of any material

Shift+Enter: Advanced          DFHack

I: Count items    R: Range
  er: Min 15    df: Max 20
A: Add limit, X: Delete

ESC: Back         S: Status
DFHack
```

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 and set the item count range.



```
*PAUSED*     FPS: 100 (49)     Dwarf Fortress                      Idlers: 9
                         New workflow constraint
                  Items matching:                  Mat class      nstraints
                                                                   (now 19)
                   Type: bars                      plant
                     t: Select, C: Crafts          wood
                                                   cloth
                   Material: any metal             silk              (now 78)
                     p: Specific                   leather
                                                   bone
                   Other:                          shell
                    -+: Ordinary quality           soap
                     l: Include foreign            tooth
                                                   horn
                                                   pearl
                                                   yarn
                  Desired range: 5-10 items        metal    √
                                                   stone
                   I: Count items     R: Range     sand        ems       R: Range
                     er: Min 5      df: Max 10     glass       5     df: Max 20
                                                   clay        t,  X: Delete
                  ESC: Cancel, y: Create new       milk
                                                                        S: Status
                                                    DFHack
                                                DFHack
```

Pressing S (or by using the hotkey in the z stocks screen) opens the overall status screen where you can manage constraints for all jobs:



```
            FPS: 100 (49)      Workflow Status

  Item          Material etc    Stock / Limit      Currently 10 (80 in use)

  any craft     iron               18  I  25       I: Count items    R: Range
  armor stand   any material        5  I  5          /*: Min 5      -+: Max 10
  barrel        any material       37  I  10
  bars          any soap            5  S  5
  bed           any wood            6  I  5
  bin           any material       10  I  10
  bolt          any bone            0  I  40
  box/bag       any cloth           0  I  10
  bucket        any wood           21  I  5
  cap           any leather         0  I  10
  cloak         any cloth           0  I  10
  coffin        any material        5  I  5
  dress         any cloth           0  I  10
  glove         any leather         0  I  20
  hood          any cloth           0  I  10
  leaves        any material        2  S  15

  any_

  A: Add, X: Delete, O: Severity Order
                                                                          DFHack
```

This screen shows all currently existing workflow constraints, and allows monitoring and/or changing them from one screen.

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).

Usage

`gui/workflow` View and manage constraints for the currently selected workshop job.

`gui/workflow status` View and manage constraints across all *workflow* managed jobs.

4.6.358 gui/workorder-details

Tags: fort | inspection | workorders

Command: `gui/workorder-details`

Adjust input materials and traits for workorders.

Keybinding: D in `workquota_details`

This tool allows you to adjust item types, materials, and/or traits for items used in manager workorders. The jobs created from those workorders will inherit the details.

Invoke while on a work order's detail screen (`j-m`, select work order, `d`).

Usage

```
gui/workorder-details
```

4.6.359 gui/workshop-job

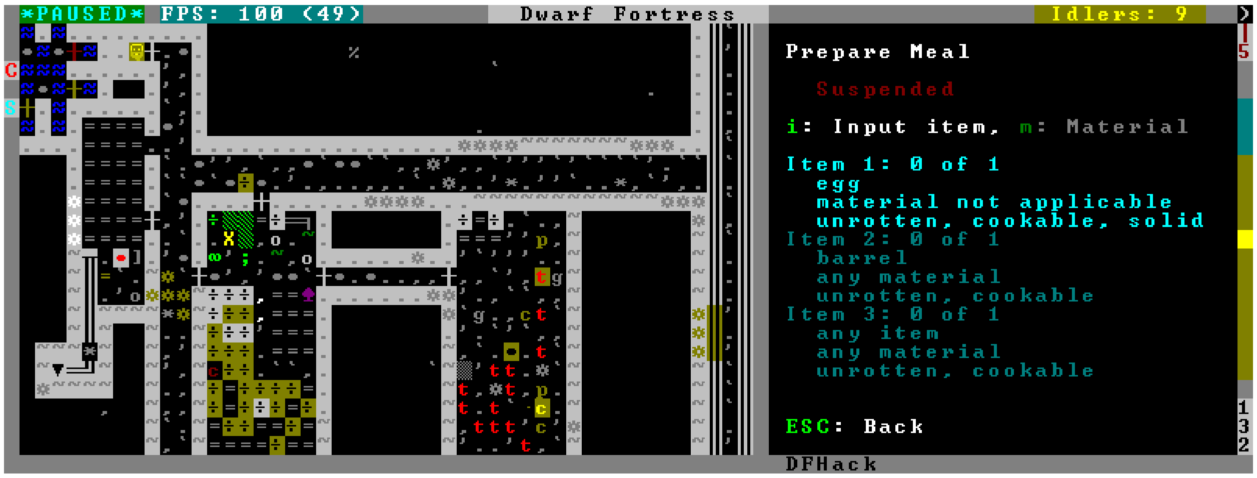
Tags: fort | inspection | jobs

Command: `gui/workshop-job`

Adjust the input materials used for a job at a workshop.

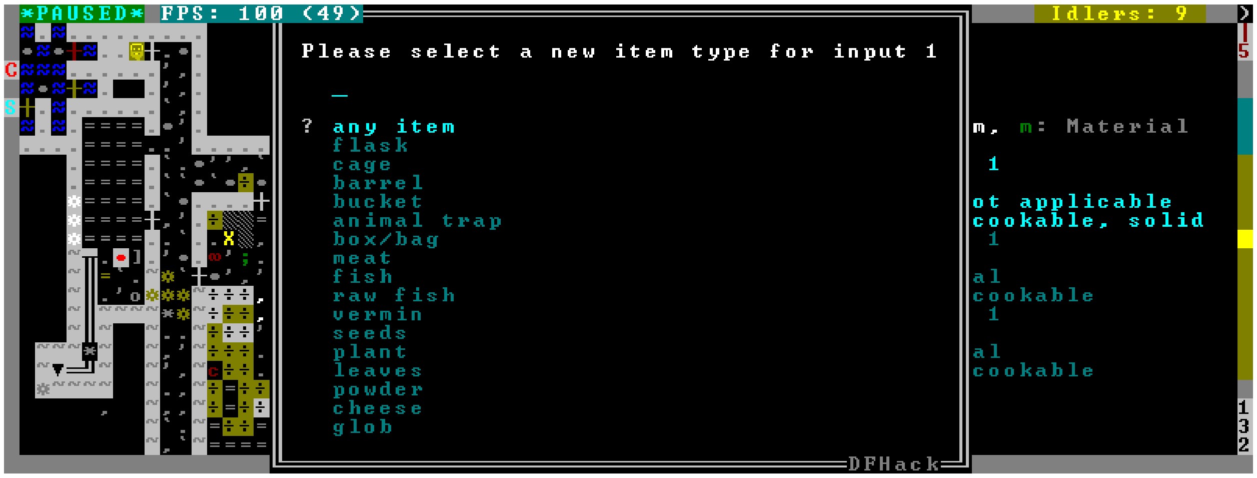
Keybinding: AltA in `dwarfmode/QueryBuilding/Some/Workshop/Job`

This tool allows you to inspect or change the input reagents for the selected workshop job (in `q` mode).



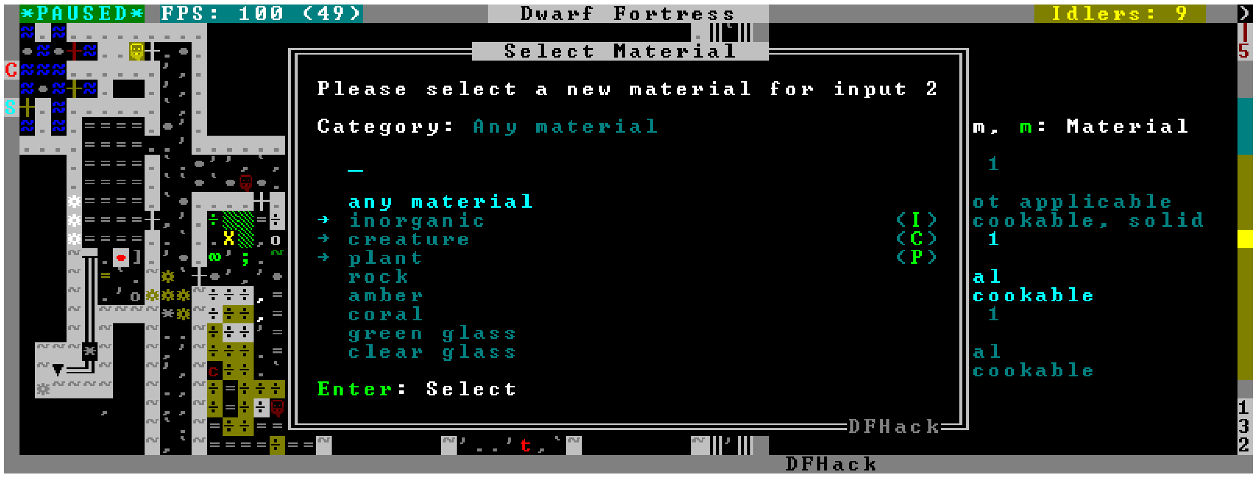
```
Prepare Meal

  Suspended

i: Input item, m: Material

Item 1: 0 of 1
  egg
  material not applicable
  unrotten, cookable, solid
Item 2: 0 of 1
  barrel
  any material
  unrotten, cookable
Item 3: 0 of 1
  any item
  any material
  unrotten, cookable

ESC: Back
```

Pressing `i` shows a dialog where you can select an item type from a list.



```
Please select a new item type for input 1

_

? any item
  flask
  cage
  barrel
  bucket
  animal trap
  box/bag
  meat
  fish
  raw fish
  vermin
  seeds
  plant
  leaves
  powder
  cheese
  glob
```

Pressing `m` (unless the item type does not allow a material) lets you choose a material.



```
Select Material

Please select a new material for input 2

Category: Any material

_

  any material
→ inorganic                          (I)
→ creature                           (C)
→ plant                              (P)
  rock
  amber
  coral
  green glass
  clear glass

Enter: Select
```

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 may be matched incorrectly. If you press `m` without choosing an item type, the script will auto-choose if there is only one valid choice.

Note that the choices presented in the dialogs are constrained by the job input flags. For example, if you choose a `plant` input item type for your `prepare meal` job, it will only let you select cookable plants since the job reagent has the `cookable` trait.

As another example, if you choose a `barrel` item for your `prepare meal` job (meaning things stored in barrels, like drink or milk), it will let you select any material that barrels can be made out of, 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, it won't let you select `plant` because plants cannot be made of iron – you have to unset the material first.

Usage

```
gui/workshop-job
```

4.6.360 modtools/add-syndrome

Tags: dev

Command: `modtools/add-syndrome`

Add and remove syndromes from units.

This allows adding and removing syndromes from units.

Usage

```
modtools/add-syndrome --target <id> --syndrome <name>|<id> [<options>]
modtools/add-syndrome --target <id> --eraseClass <class>
```

Examples

`modtools/add-syndrome --target 2391 --syndrome "gila monster bite" --eraseAll` Remove all instances of the "gila monster bite" syndrome from the specified unit.

`modtools/add-syndrome --target 1231 --syndrome 14 --resetPolicy DoNothing` Adds syndrome 14 to the specified unit, but only if that unit doesn't already have the syndrome.

Options

--target <id> The unit id of the target unit.

--syndrome <name>|<id> The syndrome to work with.

--resetPolicy <policy> Specify a policy of what to do if the unit already has an instance of the syndrome, one of: `NewInstance`, `DoNothing`, `ResetDuration`, or `AddDuration`. By default, it will create a new instance of the syndrome, even if one already exists for the unit.

--erase Instead of adding an instance of the syndrome, erase one.

--eraseAll Erase every instance of the syndrome.

--eraseClass <class id> Erase every instance of every syndrome with the given SYN_CLASS (an integer id).

--skipImmunities Add the syndrome to the target even if it is immune to the syndrome.

4.6.361 modtools/anonymous-script

Tags: dev

Command: `modtools/anonymous-script`

Run dynamically generated script code.

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"
```

4.6.362 modtools/change-build-menu

Tags: dev

Command: `modtools/change-build-menu`

Add or remove items from the build sidebar menus.

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:

`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.

`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.

4.6.363 modtools/create-item

Tags: dev

Command: `modtools/create-item`

Create arbitrary items.

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
```

(continues on next page)

(continued from previous page)

```
-matchingGloves
    create two of this item, and set handedness appropriately
```

4.6.364 modtools/create-tree

Tags: dev

Command: `modtools/create-tree`

Spawn trees.

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 ]
```

4.6.365 modtools/create-unit

Tags: dev

Command: `modtools/create-unit`

Create arbitrary units.

Creates a unit. Usage:

```
-race raceName
    (obligatory)
    Specify the race of the unit to be created.
    examples:
```

(continues on next page)

(continued from previous page)

```
        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. Can be passed
    empty to generate a wild name (random language, any words), i.e.
    the type of name that animal people historical figures have.
    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
```

(continues on next page)

(continued from previous page)

```
        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,
        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"
```

(continues on next page)

(continued from previous page)

```
-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.
    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 ]
```

(continues on next page)

(continued from previous page)

```
            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.
```

4.6.366 modtools/equip-item

Tags: dev

Command: `modtools/equip-item`

Force a unit to equip an item.

Force a unit to equip an item with a particular body part; useful in conjunction with the `create` scripts above. See also *forceequip*.

4.6.367 modtools/extra-gamelog

Tags: dev

Command: `modtools/extra-gamelog`

Write info to the gamelog for Soundsense.

This script writes extra information to the gamelog. This is useful for tools like Soundsense.

Usage:

```
modtools/extra-gamelog enable
modtools/extra-gamelog disable
```

4.6.368 modtools/fire-rate

Tags: dev

Command: `modtools/fire-rate`

Alter the fire rate of ranged weapons.

Allows altering the fire rates of ranged weapons. Each are defined on a per-item basis. As this is done in an on-world basis, commands for this should be placed in an `onLoad*.init`. This also technically serves as a patch to any of the weapons targeted in adventure mode, reducing the times down to their intended speeds (the game applies an additional hardcoded recovery time to any ranged attack you make in adventure mode).

Once run, all ranged attacks will use this script's systems for calculating recovery speeds, even for items that haven't directly been modified using this script's commands. One minor side effect is that it can't account for interactions with the `FREE_ACTION` token; interactions with that tag which launch projectiles will be subject to recovery times (though there aren't any interaction in vanilla where this would happen, as far as I know).

Requires a Target and any number of Modifiers.

Targets:

-item <item token> The full token of the item to modify. Example: `WEAPON:ITEM_WEAPON_BOW`

-throw Modify the fire rate for throwing. This is specifically for thrown attacks without a weapon - if you have a weapon that uses `THROW` as its skill, you need to use the `-item` argument for it.

Modifiers:

-material <material token> Apply only to items made of the given material token. With the `-item` argument, this will apply to the material that the weapon is made of, whereas with the `-throw` argument this will apply to the material being thrown (or fired, in the case of interactions). This is optional. Format examples: "CREATURE:COW:MILK", "PLANT:MUSHROOM_HELMET_PLUMP:DRINK", "INORGANIC:GOLD", "VOMIT"

-fortBase <integer> -advBase <integer> Set the base fire rate for the weapon in ticks to use in the respective mode (fortress/adventure). Means one shot per x ticks. Defaults to the game default of 80.

-fortSkillFactor <float> -advSkillFactor <float> Multiplier that modifies how effective a user's skill is at improving the fire rate in the respective modes. In basic mode, recovery time is reduced by this value * user's skill ticks. Defaults to 2.7. With that value and default settings, it will make a Legendary shooter fire at the speed cap.

-fortCap <integer> -advCap <integer> Sets a cap on the fastest fire rate that can be achieved in their respective mode. Due to game limitations, the cap can't be less than 10 in adventure mode. Defaults to half of the base fire rate defined by the `-fort` or `-adv` arguments.

Other:

-mode <"basic" | "vanilla"> Sets what method is used to determine how skill affects fire rates. This is applied globally, rather than on a per-item basis. Basic uses a simplified method for working out fire rates - each point in a skill reduces the fire cooldown by a consistent, fixed amount. This method is the default. Vanilla mode attempts to replicate behaviour for fire rates - skill rolls determine which of 6 fixed increments of speeds is used, with a unit's skill affecting the range and averages. **NOT YET IMPLEMENTED!**

4.6.369 modtools/force

Tags: dev

Command: `modtools/force`

Trigger game events.

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
```

4.6.370 modtools/if-entity

Tags: dev

Command: `modtools/if-entity`

Run DFHack commands based on current civ id.

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.')" ]
  ```

4.6.371 modtools/interaction-trigger

Tags: dev

Command: `modtools/interaction-trigger`

Run DFHack commands when a unit attacks or defends.

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.

4.6.372 modtools/invader-item-destroyer

Tags: dev

Command: `modtools/invader-item-destroyer`

Destroy invader items when they die.

This tool can destroy 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:
        MOUNTAIN
        EVIL
```

4.6.373 modtools/item-trigger

Tags: dev

Command: `modtools/item-trigger`

Run DFHack commands when a unit uses an item.

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
```

(continues on next page)

(continued from previous page)

```
    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 command 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
        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
```

4.6.374 modtools/moddable-gods

Tags: dev

Command: `modtools/moddable-gods`

Create deities.

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
```

4.6.375 modtools/outside-only

Tags: dev

Command: `modtools/outside-only`

Set building inside/outside restrictions.

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
```

4.6.376 modtools/pref-edit

Tags: dev

Command: `modtools/pref-edit`

Modify unit preferences.

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
```

4.6.377 modtools/projectile-trigger

Tags: dev

Command: `modtools/projectile-trigger`

Run DFHack commands when projectiles hit their targets.

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 ]
    \\LOCATION
    \\PROJECTILE_ID
    \\FIRER_ID
    \\anything -> \anything
    anything -> anything
```

4.6.378 modtools/random-trigger

Tags: dev

Command: `modtools/random-trigger`

Randomly select DFHack scripts to run.

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.

4.6.379 modtools/raw-lint

Tags: dev

Command: `modtools/raw-lint`

Check for errors in raw files.

Checks for simple issues with raw files. Can be run automatically.

4.6.380 modtools/reaction-product-trigger

Tags: dev

Command: `modtools/reaction-product-trigger`

Call DFHack commands when reaction products are produced.

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
```

4.6.381 modtools/reaction-trigger

Tags: dev

Command: `modtools/reaction-trigger`

Run DFHack commands when custom reactions complete.

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
```

(continues on next page)

(continued from previous page)

```
    if absent:
        if running a script, only one target will be used
        if applying a syndrome, then only one target will be infected
-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 eligible 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
```

4.6.382 modtools/reaction-trigger-transition

Tags: dev

Command: `modtools/reaction-trigger-transition`

Help create reaction triggers.

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.

4.6.383 modtools/set-belief

Tags: dev

Command: `modtools/set-belief`

Change the beliefs/values of a unit.

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:

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 *.

4.6.384 modtools/set-need

Tags: dev

Command: `modtools/set-need`

Change the needs of a unit.

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
```

4.6.385 modtools/set-personality

Tags: dev

Command: `modtools/set-personality`

Change a unit's personality.

Changes the personality of units.

Usage

```
modtools/set-personality --list
modtools/set-personality [<target option>] <trait option> <modifier option>
```

If no target option is given, the unit selected in the UI is used by default.

Target options

`--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.

Trait options

`--all` Apply the edit to all the target's traits.

`--trait <ID>` ID of the trait to edit. For example, 0 or HATE_PROPENSITY.

Modifier options

`--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).

Other options

--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.

4.6.386 modtools/skill-change

Tags: dev

Command: `modtools/skill-change`

Modify unit skills.

Sets or modifies a skill of a unit.

Usage

```
modtools/skill-change --unit <id> --skill <skill> --mode <mode> --granularity
↪<granularity> --value <amount> [--loud]
```

Options

--unit <id> Id of the target unit.

--skill <skill> Specify which skill to set.

--mode <mode> Mode can be `add` or `set`, depending on whether you want to add to the existing experience/level or set it.

--granularity <granularity> Granularity can be `experience` or `level`, depending on whether you want to modify/set the experience value or the experience level.

--value <amount> How much to set/add.

--loud if present, prints changes to console

4.6.387 modtools/spawn-flow

Tags: dev

Command: `modtools/spawn-flow`

Creates flows at the specified location.

Creates flows at the specified location.

Usage

```
modtools/spawn-flow --material <TOKEN> --flowType <type> --location [ <x> <y> <z> ] [--
↪flowSize <size>]
```

Options

`--material <TOKEN>` Specify the material of the flow, if applicable. E.g. `INORGANIC:IRON`, `CREATURE_MAT:DWARF:BRAIN`, or `PLANT_MAT:MUSHROOM_HELMET_PLUMP:DRINK`.

`--flowType <type>` The flow type, one of:

```
Miasma
Steam
Mist
MaterialDust
MagmaMist
Smoke
Dragonfire
Fire
Web
MaterialGas
MaterialVapor
OceanWave
SeaFoam
```

`--location [ <x> <y> <z> ]` The location to spawn the flow

`--flowSize <size>` Specify how big the flow is (default: 100).

4.6.388 modtools/spawn-liquid

Tags: dev

Command: `modtools/spawn-liquid`

Spawn water or lava.

This script spawns liquid at the given coordinates.

Usage

```
modtools/spawn-liquid <height> water|magma <x> <y> <z> [<xOff> <yOff> <zOff>]
```

Options

`<height>` The height of the water/magma (1 to 7)

`<x>`, `<y>`, `<z>` The location to spawn liquid at (replacing any preexisting liquid).

`<xOff>`, `<yOff>`, `<zOff>` Optional convenience offsets, added to <x>, <y>, and <z>.

4.6.389 modtools/syndrome-trigger

Tags: dev

Command: `modtools/syndrome-trigger`

Trigger DFHack commands when units acquire syndromes.

This script helps you set up commands that trigger when syndromes are applied to units.

Usage

```
modutils/syndrome-trigger --clear
modutils/syndrome-trigger --syndrome <name> --command [ <command> ]
modutils/syndrome-trigger --synclass <class> --command [ <command> ]
```

Options

`--clear` Clear any previously registered syndrome triggers.

`--syndrome <name>` Specify a syndrome by its name. Enclose the name in quotation marks if it includes spaces (e.g. `--syndrome "gila monster bite"`).

`--synclass <class>` Any syndrome with the specified SYN_CLASS will act as a trigger. Enclose in quotation marks if it includes spaces.

`--command [ <command> ]` Specify the command to be executed after infection. Remember to include a space before and after the square brackets! The following tokens 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.

Examples

```
modutils/syndrome-trigger --synclass VAMPCURSE --command [ modtools/spawn-flow -flowType␣
↪Dragonfire -location [ \\LOCATION ] ]
```

4.6.390 modtools/transform-unit

Tags: dev

Command: `modtools/transform-unit`

Transform a unit into another unit type.

This tool transforms a unit into another unit type, either temporarily or 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.

Usage

```
modtools/transform-unit --unit <id> --race <race> --caste <caste> [--duration <ticks>] [-
↪-keepInventory] [--setPrevRace]
modtools/transform-unit --unit <id> --untransform
modtools/transform-unit --clear
```

Options

`--unit <id>` Set the target unit.

`--race <race>` Set the target race.

`--caste <caste>` Set the target caste.

`--duration <ticks>` Set how long the transformation should last, or "forever". If not specified, then the transformation is permanent.

`--keepInventory` Move items back into inventory after transformation

`--setPrevRace` Remember the previous race so that you can change the unit back with `--untransform`

`--untransform` Turn the unit back into what it was before (assuming you used the `--setPrevRace` option when transforming the first time).

`--clear` Clear records of "previous" races used by the `--untransform` option.

CHAPTER

FIVE

USER GUIDES

These pages are detailed guides covering DFHack tools.

5.1 DFHack modding guide

5.1.1 What is the difference between a script and a mod?

A script is a single file that can be run as a command in DFHack, like something that modifies or displays game data on request. A mod is something you install to get persistent behavioural changes in the game and/or add new content. Mods can contain and use scripts in addition to (or instead of) modifications to the DF game raws.

DFHack scripts are written in Lua. If you don't already know Lua, there's a great primer at lua.org.

5.1.2 Why not just mod the raws?

It depends on what you want to do. Some mods *are* better to do in just the raws. You don't need DFHack to add a new race or modify attributes, for example. However, DFHack scripts can do many things that you just can't do in the raws, like make a creature that trails smoke. Some things *could* be done in the raws, but writing a script is less hacky, easier to maintain, easier to extend, and is not prone to side-effects. A great example is adding a syndrome when a reaction is performed. If done in the raws, you have to create an exploding boulder to apply the syndrome. DFHack scripts can add the syndrome directly and with much more flexibility. In the end, complex mods will likely require a mix of raw modding and DFHack scripting.

5.1.3 A mod-maker's development environment

While you're writing your mod, you need a place to store your in-development scripts that will:

- be directly runnable by DFHack
- not get lost when you upgrade DFHack

The recommended approach is to create a directory somewhere outside of your DF installation (let's call it "/path/to/own-scripts") and do all your script development in there.

Inside your DF installation folder, there is a file named `dfhack-config/script-paths.txt`. If you add a line like this to that file:

```
+/path/to/own-scripts
```

Then that directory will be searched when you run DFHack commands from inside the game. The + at the front of the path means to search that directory first, before any other script directory (like `hack/scripts` or `raw/scripts`). That way, your latest changes will always be used instead of older copies that you may have installed in a DF directory.

For scripts with the same name, the *order of precedence* will be:

1. `own-scripts/`
2. `data/save/*/raw/scripts/`
3. `raw/scripts/`
4. `hack/scripts/`

5.1.4 The structure of the game

"The game" is in the global variable *df*. The game's memory can be found in `df.global`, containing things like the list of all items, whether to reindex pathfinding, et cetera. Also relevant to us in `df` are the various types found in the game, e.g. `df.pronoun_type` which we will be using in this guide. We'll explore more of the game structures below.

5.1.5 Your first script

So! It's time to write your first script. This section will walk you through how to make a script that will get the pronoun type of the currently selected unit.

First line, we get the unit:

```
local unit = dfhack.gui.getSelectedUnit()
```

If no unit is selected, an error message will be printed (which can be silenced by passing `true` to `getSelectedUnit`) and `unit` will be `nil`.

If `unit` is `nil`, we don't want the script to run anymore:

```
if not unit then
    return
end
```

Now, the field `sex` in a unit is an integer, but each integer corresponds to a string value ("it", "she", or "he"). We get this value by indexing the bidirectional map `df.pronoun_type`. Indexing the other way, incidentally, with one of the strings, will yield its corresponding number. So:

```
local pronounTypeString = df.pronoun_type[unit.sex]
print(pronounTypeString)
```

Simple. Save this as a Lua file in your own scripts directory and run it as shown before when a unit is selected in the Dwarf Fortress UI.

5.1.6 Exploring DF structures

So how could you have known about the field and type we just used? Well, there are two main tools for discovering the various fields in the game's data structures. The first is the `df-structures` repository that contains XML files describing the contents of the game's structures. These are complete, but difficult to read (for a human). The second option is the *gui/gm-editor* script, an interactive data explorer. You can run the script while objects like units are selected to view the data within them. You can also run `gui/gm-editor scr` to view the data for the current screen. Press `?` while the script is active to view help.

Familiarising yourself with the many structs of the game will help with ideas immensely, and you can always ask for help in the *right places*.

5.1.7 Detecting triggers

The common method for injecting new behaviour into the game is to define a callback function and get it called when something interesting happens. DFHack provides two libraries for this, `repeat-util` and *eventful*. `repeat-util` is used to run a function once per a configurable number of frames (paused or unpaused), ticks (unpaused), in-game days, months, or years. If you need to be aware the instant something happens, you'll need to run a check once a tick. Be careful not to do this gratuitously, though, since running that often can slow down the game!

`eventful`, on the other hand, is much more performance-friendly since it will only call your callback when a relevant event happens, like a reaction or job being completed or a projectile moving.

To get something to run once per tick, we can call `repeat-util.scheduleEvery()`. First, we load the module:

```lua
local repeatUtil = require('repeat-util')
```

Both `repeat-util` and `eventful` require keys for registered callbacks. You should use something unique, like your mod name:

```lua
local modId = "callback-example-mod"
```

Then, we pass the key, amount of time units between function calls, what the time units are, and finally the callback function itself:

```lua
repeatUtil.scheduleEvery(modId, 1, "ticks", function()
    -- Do something like iterating over all active units and
    -- check for something interesting
    for _, unit in ipairs(df.global.world.units.active) do
        ...
    end
end)
```

`eventful` is slightly more involved. First get the module:

```lua
local eventful = require('plugins.eventful')
```

`eventful` contains a table for each event which you populate with functions. Each function in the table is then called with the appropriate arguments when the event occurs. So, for example, to print the position of a moving (item) projectile:

```lua
eventful.onProjItemCheckMovement[modId] = function(projectile)
    print(projectile.cur_pos.x, projectile.cur_pos.y,
          projectile.cur_pos.z)
end
```

Check out the *full list of supported events* to see what else you can react to with `eventful`.

Now, you may have noticed that you won't be able to register multiple callbacks with a single key named after your mod. You can, of course, call all the functions you want from a single registered callback. Alternately, you can create multiple callbacks using different keys, using your mod ID as a key name prefix. If you do register multiple callbacks, though, there are no guarantees about the call order.

5.1.8 Custom raw tokens

In this section, we are going to use *custom raw tokens* applied to a reaction to transfer the material of a reagent to a product as a handle improvement (like on artifact buckets), and then we are going to see how you could make boots that make units go faster when worn.

First, let's define a custom crossbow with its own custom reaction. The crossbow:

```
[ITEM_WEAPON:ITEM_WEAPON_CROSSBOW_SIEGE]
    [NAME:crossbow:crossbows]
    [SIZE:600]
    [SKILL:HAMMER]
    [RANGED:CROSSBOW:BOLT]
    [SHOOT_FORCE:4000]
    [SHOOT_MAXVEL:800]
    [TWO_HANDED:0]
    [MINIMUM_SIZE:17500]
    [MATERIAL_SIZE:4]
    [ATTACK:BLUNT:10000:4000:bash:bashes:NO_SUB:1250]
        [ATTACK_PREPARE_AND_RECOVER:3:3]
    [SIEGE_CROSSBOW_MOD_FIRE_RATE_MULTIPLIER:2] custom token (you'll see)
```

The reaction to make it (you would add the reaction and not the weapon to an entity raw):

```
[REACTION:MAKE_SIEGE_CROSSBOW]
    [NAME:make siege crossbow]
    [BUILDING:BOWYER:NONE]
    [SKILL:BOWYER]
    [REAGENT:mechanism 1:2:TRAPPARTS:NONE:NONE:NONE]
    [REAGENT:bar:150:BAR:NONE:NONE:NONE]
        [METAL_ITEM_MATERIAL]
    [REAGENT:handle 1:1:BLOCKS:NONE:NONE:NONE] wooden handles
        [ANY_PLANT_MATERIAL]
    [REAGENT:handle 2:1:BLOCKS:NONE:NONE:NONE]
        [ANY_PLANT_MATERIAL]
    [SIEGE_CROSSBOW_MOD_TRANSFER_HANDLE_MATERIAL_TO_PRODUCT_IMPROVEMENT:1]
        another custom token
    [PRODUCT:100:1:WEAPON:ITEM_WEAPON_CROSSBOW_SIEGE:GET_MATERIAL_FROM_REAGENT:bar:NONE]
```

So, we are going to use the `eventful` module to make it so that (after the script is run) when this crossbow is crafted, it will have two handles, each with the material given by the block reagents.

First, require the modules we are going to use:

```lua
local eventful = require("plugins.eventful")
local customRawTokens = require("custom-raw-tokens")
```

Now, let's make a callback (we'll be defining the body of this function soon):

```
local modId = "siege-crossbow-mod"
eventful.onReactionComplete[modId] = function(reaction,
        reactionProduct, unit, inputItems, inputReagents,
        outputItems)
```

First, we check to see if it the reaction that just happened is relevant to this callback:

```
if not customRawTokens.getToken(reaction,
    "SIEGE_CROSSBOW_MOD_TRANSFER_HANDLE_MATERIAL_TO_PRODUCT_IMPROVEMENT")
then
    return
end
```

Then, we get the product number listed. Next, for every reagent, if the reagent name starts with "handle" then we get the corresponding item, and…

```
for i, reagent in ipairs(inputReagents) do
    if reagent.code:startswith('handle') then
        -- Found handle reagent
        local item = inputItems[i]
```

…We then add a handle improvement to the listed product within our loop:

```
local new = df.itemimprovement_itemspecificst:new()
new.mat_type, new.mat_index = item.mat_type, item.mat_index
new.type = df.itemimprovement_specific_type.HANDLE
outputItems[productNumber - 1].improvements:insert('#', new)
```

This works well as long as you don't have multiple stacks filling up one reagent.

Let's also make some code to modify the fire rate of our siege crossbow:

```
eventful.onProjItemCheckMovement[modId] = function(projectile)
    if projectile.distance_flown > 0 then
        -- don't make this adjustment more than once
        return
    end

    local firer = projectile.firer
    if not firer then
        return
    end

    local weapon = df.item.find(projectile.bow_id)
    if not weapon then
        return
    end

    local multiplier = tonumber(customRawTokens.getToken(
            weapon.subtype,
            "SIEGE_CROSSBOW_MOD_FIRE_RATE_MULTIPLIER")) or 1
    firer.counters.think_counter = math.floor(
            firer.counters.think_counter * multiplier)
end
```

Now, let's see how we could make some "pegasus boots". First, let's define the item in the raws:

```
[ITEM_SHOES:ITEM_SHOES_BOOTS_PEGASUS]
    [NAME:pegasus boot:pegasus boots]
    [ARMORLEVEL:1]
    [UPSTEP:1]
    [METAL_ARMOR_LEVELS]
    [LAYER:OVER]
    [COVERAGE:100]
    [LAYER_SIZE:25]
    [LAYER_PERMIT:15]
    [MATERIAL_SIZE:2]
    [METAL]
    [LEATHER]
    [HARD]
    [PEGASUS_BOOTS_MOD_FOOT_MOVEMENT_TIMER_REDUCTION_PER_TICK:2] custom raw token
        (you don't have to comment the custom token every time,
        but it does clarify what it is)
```

Then, let's make a `repeat-util` callback for once a tick:

```
repeatUtil.scheduleEvery(modId, 1, "ticks", function()
```

Let's iterate over every active unit, and for every unit, iterate over their worn items to calculate how much we are going to take from their on-foot movement timers:

```
for _, unit in ipairs(df.global.world.units.active) do
    local amount = 0
    for _, entry in ipairs(unit.inventory) do
        if entry.mode == df.unit_inventory_item.T_mode.Worn then
            local reduction = customRawTokens.getToken(
                    entry.item,
                    'PEGASUS_BOOTS_MOD_FOOT_MOVEMENT_TIMER_REDUCTION_PER_TICK')
            amount = amount + (tonumber(reduction) or 0)
        end
    end
    -- Subtract amount from on-foot movement timers if not on ground
    if not unit.flags1.on_ground then
        dfhack.units.subtractActionTimers(unit, amount, df.unit_action_type_group.
↪MovementFeet)
    end
end
```

5.1.9 The structure of a full mod

For reference, Tachy Guns is a full mod that conforms to this guide.

Create a folder for mod projects somewhere outside your Dwarf Fortress installation directory (e.g. `/path/to/mymods/`) and use your mod IDs as the names for the mod folders within it. In the example below, we'll use a mod ID of `example-mod`. I'm sure your mods will have more creative names! The `example-mod` mod will be developed in the `/path/to/mymods/example-mod/` directory and has a basic structure that looks like this:

```
raw/init.d/example-mod.lua
raw/objects/...
raw/scripts/example-mod.lua
raw/scripts/example-mod/...
README.md
```

Let's go through that line by line.

- A short (one-line) script in `raw/init.d/` to initialise your mod when a save is loaded.
- Modifications to the game raws (potentially with custom raw tokens) go in `raw/objects/`.
- A control script in `raw/scripts/` that handles enabling and disabling your mod.
- A subfolder for your mod under `raw/scripts/` will contain all the internal scripts and/or modules used by your mod.

It is a good idea to use a version control system to organize changes to your mod code. You can create a separate Git repository for each of your mods. The `README.md` file will be your mod help text when people browse to your online repository.

Unless you want to install your `raw/` folder into your DF game folder every time you make a change to your scripts, you should add your development scripts directory to your script paths in `dfhack-config/script-paths.txt`:

```
+/path/to/mymods/example-mod/raw/scripts/
```

Ok, you're all set up! Now, let's take a look at an example `raw/scripts/example-mod.lua` file:

```
-- main setup and teardown for example-mod
-- this next line indicates that the script supports the "enable"
-- API so you can start it by running "enable example-mod" and stop
-- it by running "disable example-mod"
--@ enable = true

local usage = [[
Usage
-----

    enable example-mod
    disable example-mod
]]
local repeatUtil = require('repeat-util')
local eventful = require('plugins.eventful')

-- you can reference global values or functions declared in any of
-- your internal scripts
local moduleA = reqscript('example-mod/module-a')
local moduleB = reqscript('example-mod/module-b')
local moduleC = reqscript('example-mod/module-c')
local moduleD = reqscript('example-mod/module-d')

enabled = enabled or false
local modId = 'example-mod'

if not dfhack_flags.enable then
    print(usage)
```

(continues on next page)

(continued from previous page)

```
    print()
    print(('Example mod is currently '):format(
            enabled and 'enabled' or 'disabled'))
    return
end

if dfhack_flags.enable_state then
    -- do any initialization your internal scripts might require
    moduleA.onLoad()
    moduleB.onLoad()

    -- multiple functions in the same repeat callback
    repeatUtil.scheduleEvery(modId .. ' every tick', 1, 'ticks', function()
        moduleA.every1Tick()
        moduleB.every1Tick()
    end)

    -- one function per repeat callback (you can put them in the
    -- above format if you prefer)
    repeatUtil.scheduleEvery(modId .. ' 100 frames', 1, 'frames',
                             moduleD.every100Frames)

    -- multiple functions in the same eventful callback
    eventful.onReactionComplete[modId] = function(reaction,
            reaction_product, unit, input_items, input_reagents,
            output_items)
        -- pass the event's parameters to the listeners
        moduleB.onReactionComplete(reaction, reaction_product,
                unit, input_items, input_reagents, output_items)
        moduleC.onReactionComplete(reaction, reaction_product,
                unit, input_items, input_reagents, output_items)
    end

    -- one function per eventful callback (you can put them in the
    -- above format if you prefer)
    eventful.onProjItemCheckMovement[modId] = moduleD.onProjItemCheckMovement
    eventful.onProjUnitCheckMovement[modId] = moduleD.onProjUnitCheckMovement

    print('Example mod enabled')
    enabled = true
else
    -- call any shutdown functions your internal scripts might require
    moduleA.onUnload()

    repeatUtil.cancel(modId .. ' every ticks')
    repeatUtil.cancel(modId .. ' 100 frames')

    eventful.onReactionComplete[modId] = nil
    eventful.onProjItemCheckMovement[modId] = nil
    eventful.onProjUnitCheckMovement[modId] = nil

    print('Example mod disabled')
```

(continues on next page)

(continued from previous page)

```
    enabled = false
end
```

You can call `enable example-mod` and `disable example-mod` yourself while developing, but for end users you can start your mod automatically from `raw/init.d/example-mod.lua`:

```
dfhack.run_command('enable example-mod')
```

Inside `raw/scripts/example-mod/module-a.lua` you could have code like this:

```
--@ module = true
-- The above line is required for reqscript to work

function onLoad() -- global variables are exported
    -- do initialization here
end

-- this is an internal function: local functions/variables
-- are not exported
local function usedByOnTick(unit)
    -- ...
end

function onTick() -- exported
    for _,unit in ipairs(df.global.world.units.all) do
        usedByOnTick(unit)
    end
end
```

The *reqscript* function reloads scripts that have changed, so you can modify your scripts while DF is running and just disable/enable your mod to load the changes into your ongoing game!

5.2 DFHack config file examples

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.

5.2.1 The `init/` subfolder

The init/ subfolder contains useful DFHack *Init files* that you can copy into your `dfhack-config/init` 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 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 protect 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*.

5.3 Quickfort blueprint library

This guide contains a high-level overview of the blueprints available in the quickfort blueprint library.

Each file is hyperlinked to its online version so you can see exactly what the blueprints do before you run them. Also, if you use *gui/quickfort*, you will get a live preview of which tiles will be modified by the blueprint before you apply it to your map.

5.3.1 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 `gui/quickfort dreamfort notes`.

Dreamfort blueprints are available for easy viewing and copying online.

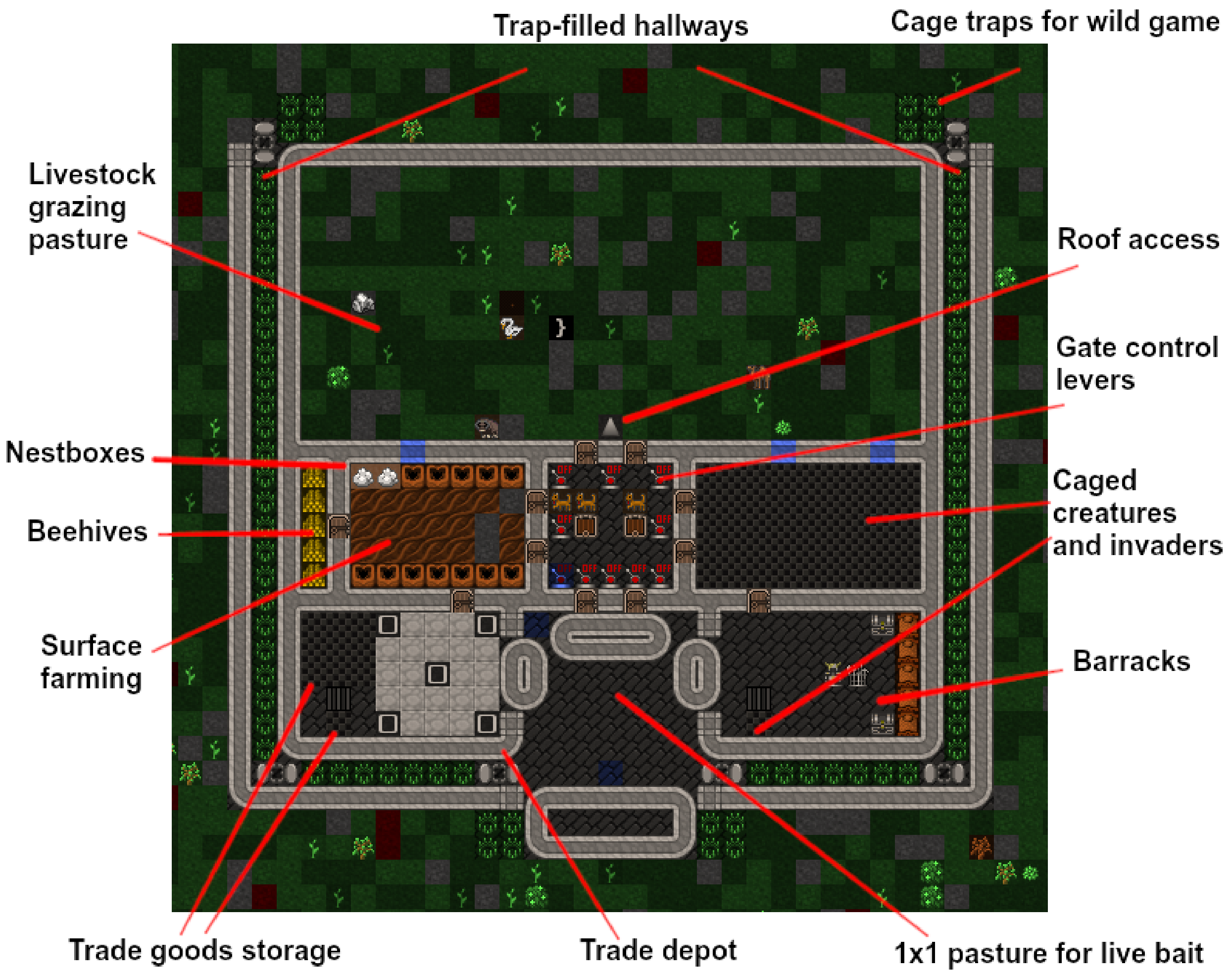
The online spreadsheets also include embark profile suggestions, a complete example embark profile, and a convenient checklist from which you can copy the `quickfort` commands.

If you like, you can download a fully built Dreamfort-based fort from dffd, load it, and explore it interactively.

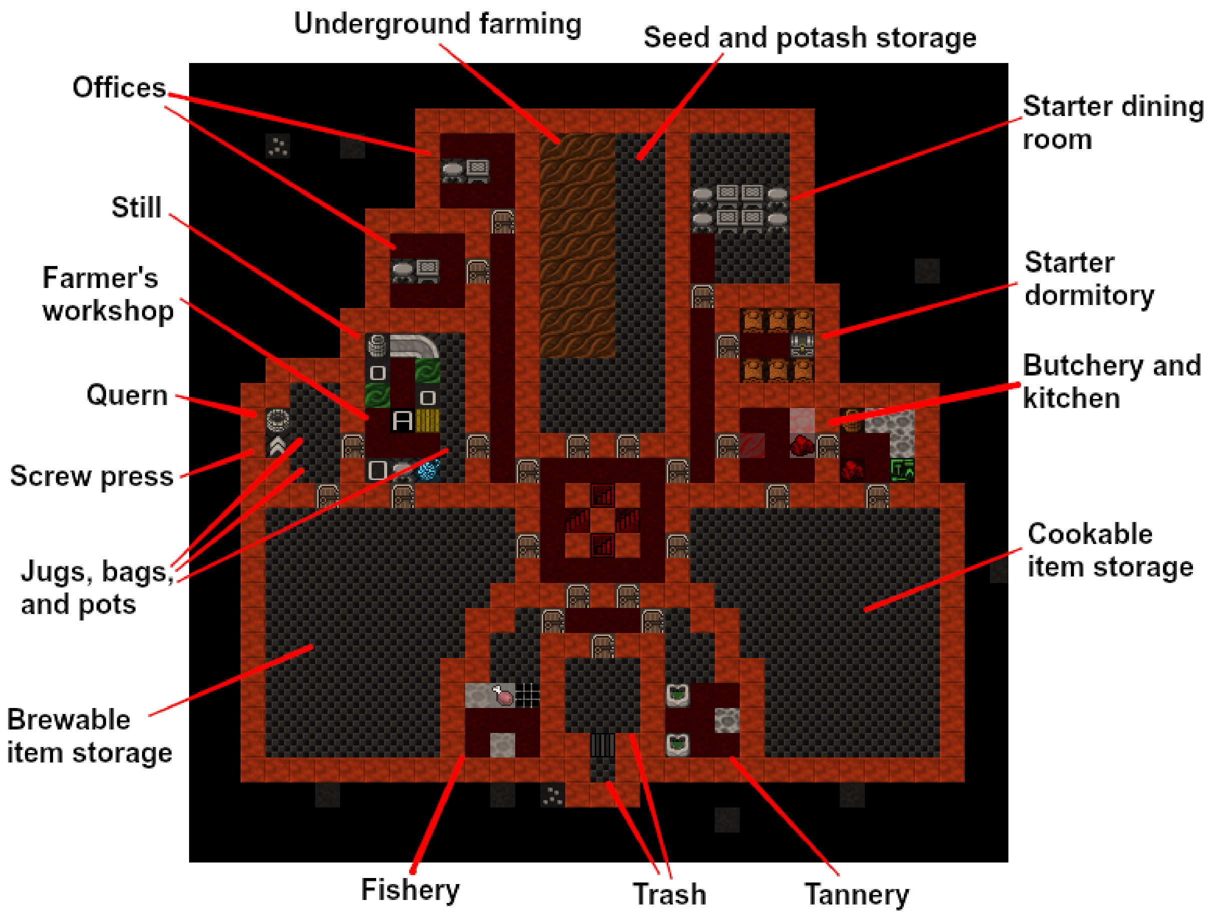
Visual overview

Here are annotated screenshots of the major Dreamfort levels (or click here for a slideshow).

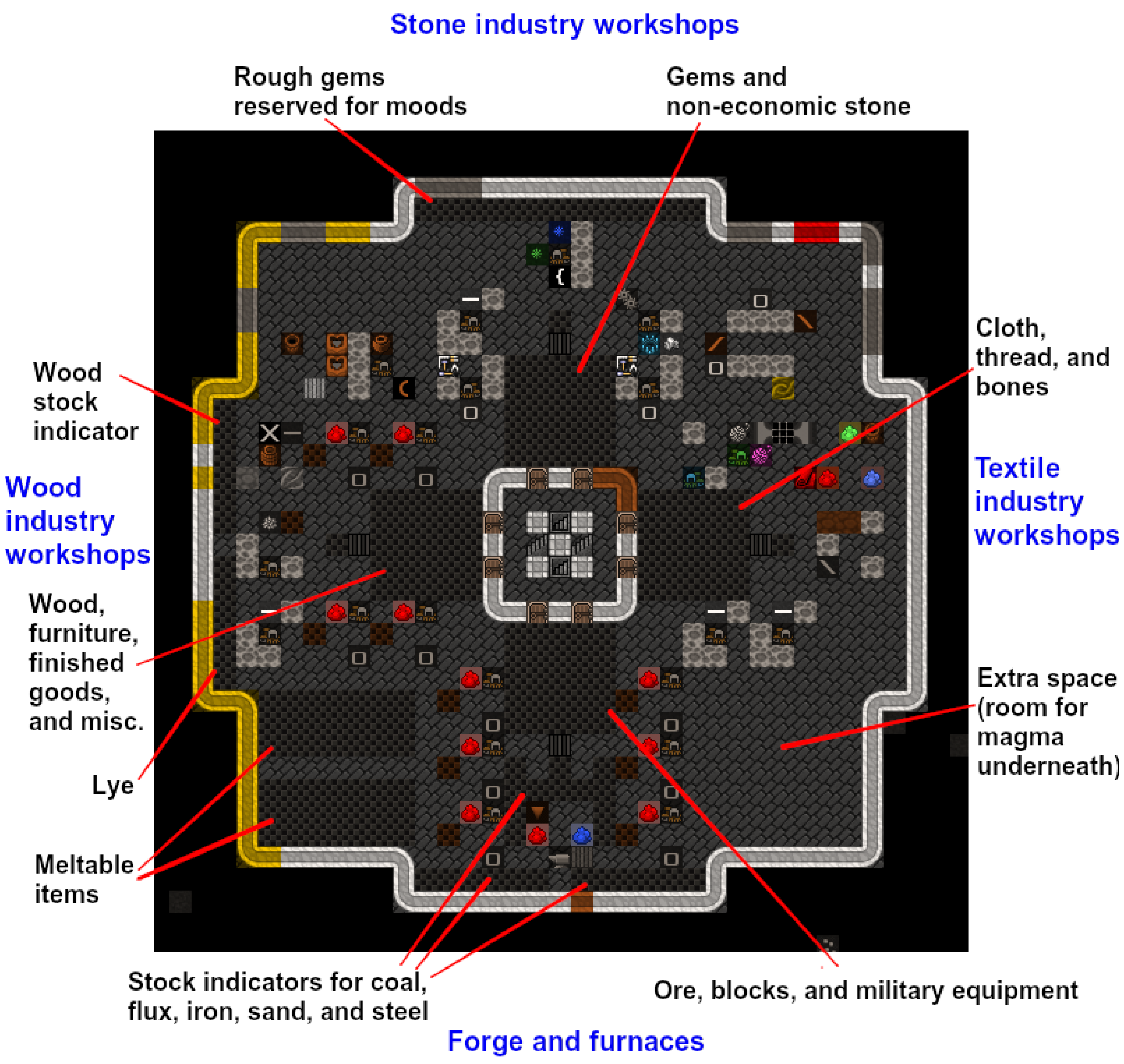
Surface level



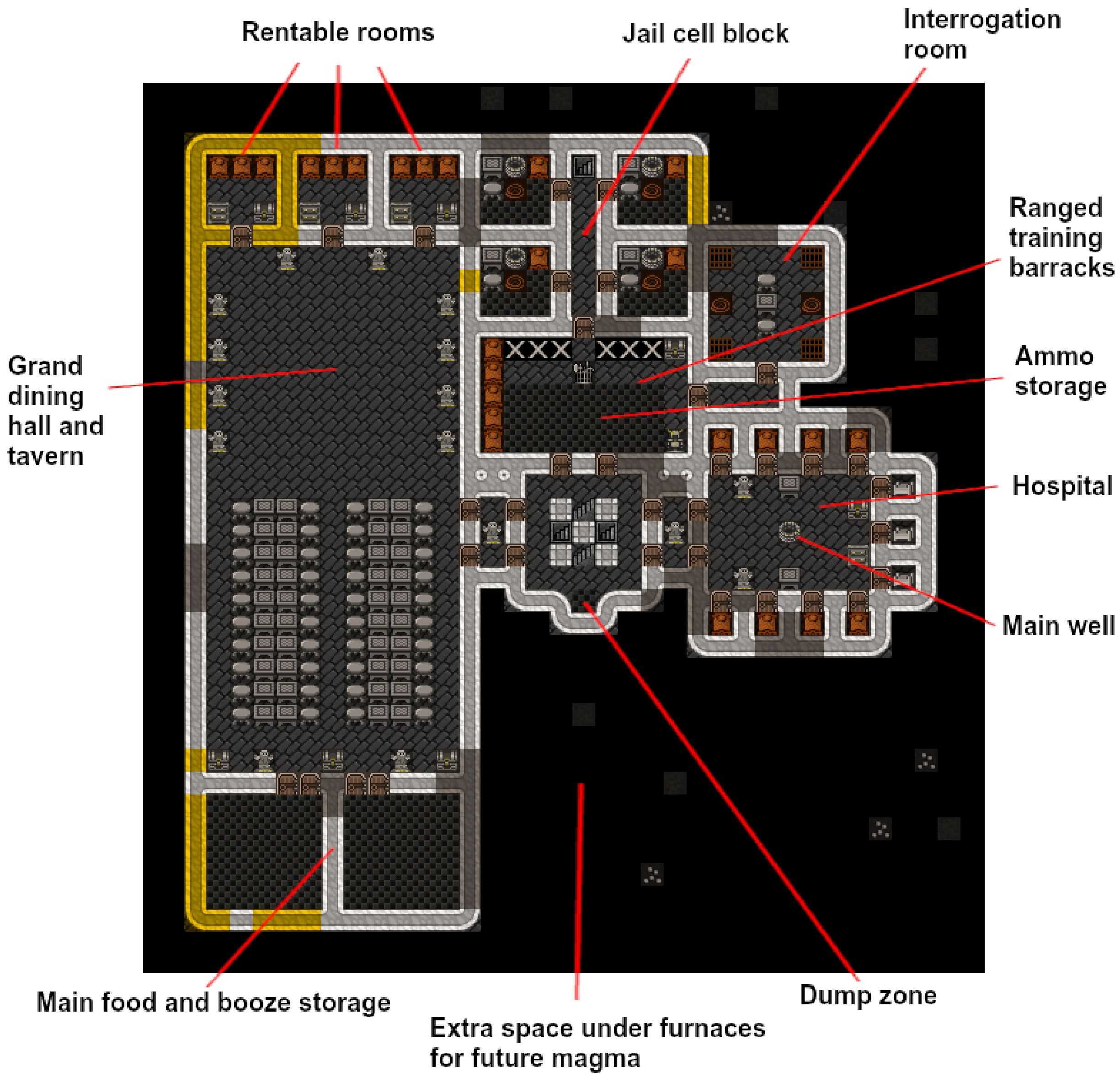
Farming level

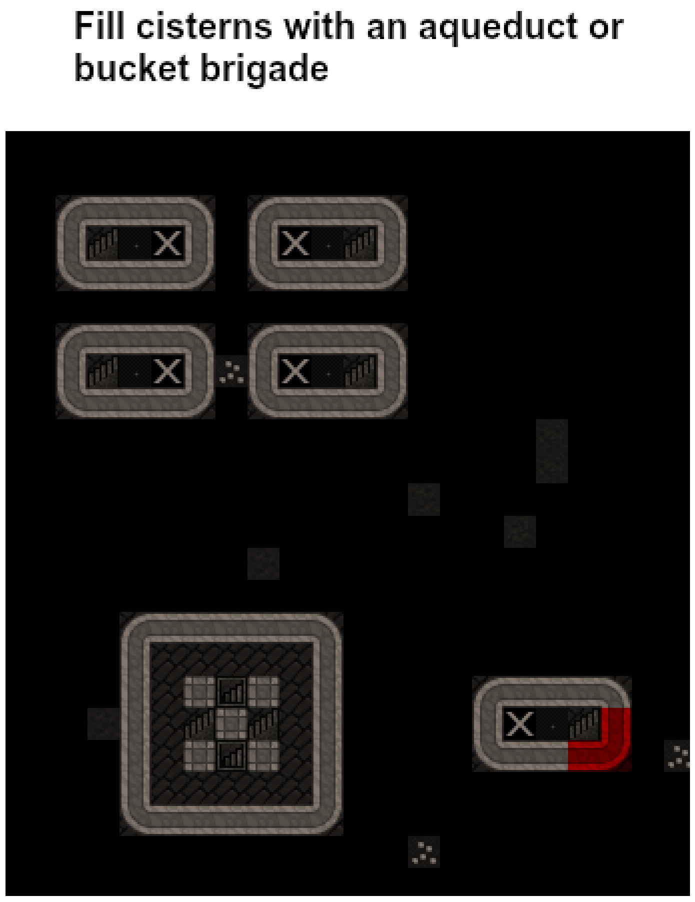


Industry level

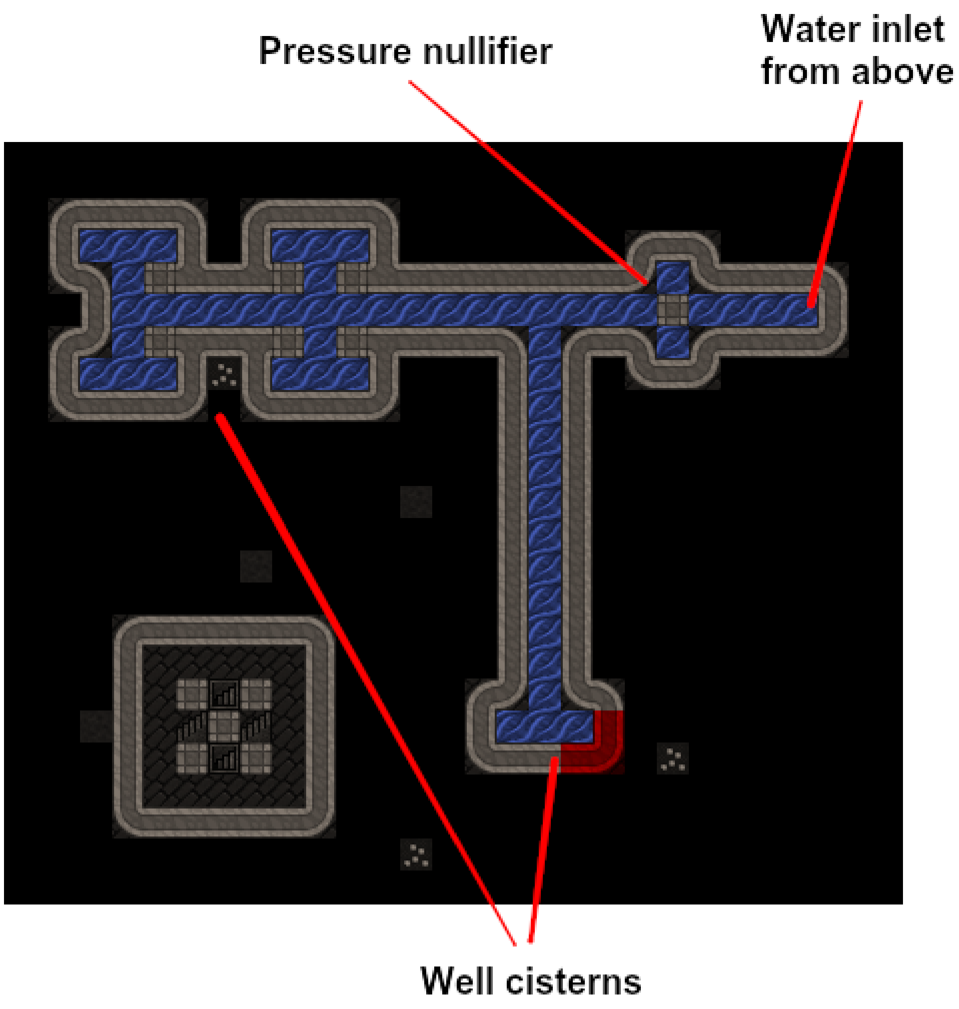

Stone industry workshops
Rough gems reserved for moods
Gems and non-economic stone
Cloth, thread, and bones
Textile industry workshops
Extra space (room for magma underneath)
Wood stock indicator
Wood industry workshops
Wood, furniture, finished goods, and misc.
Lye
Meltable items
Stock indicators for coal, flux, iron, sand, and steel
Ore, blocks, and military equipment
Forge and furnaces

Services level



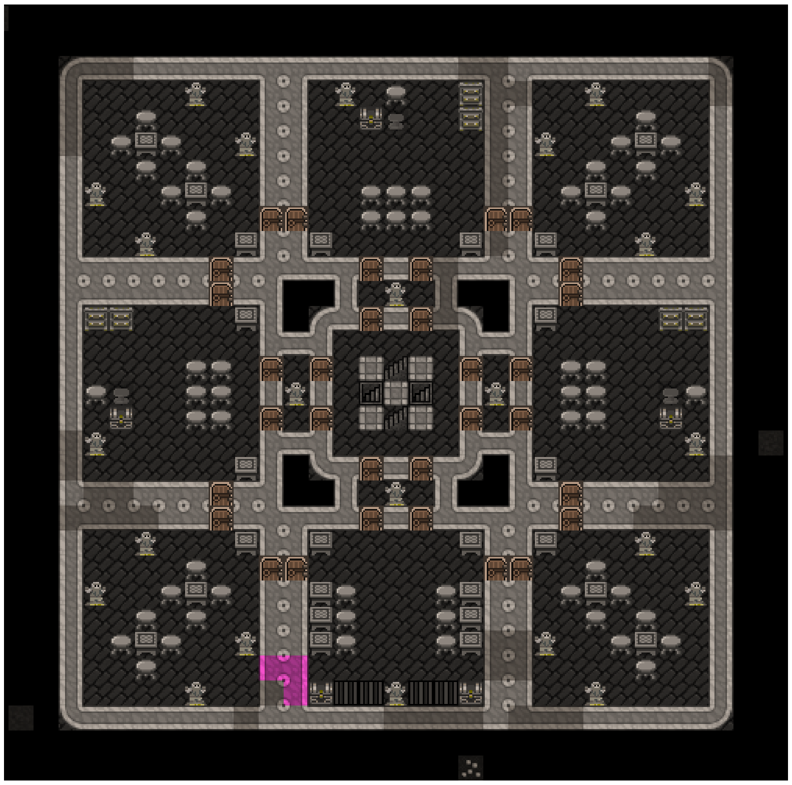

Fill cisterns with an aqueduct or bucket brigade

Example plumbing to fill cisterns



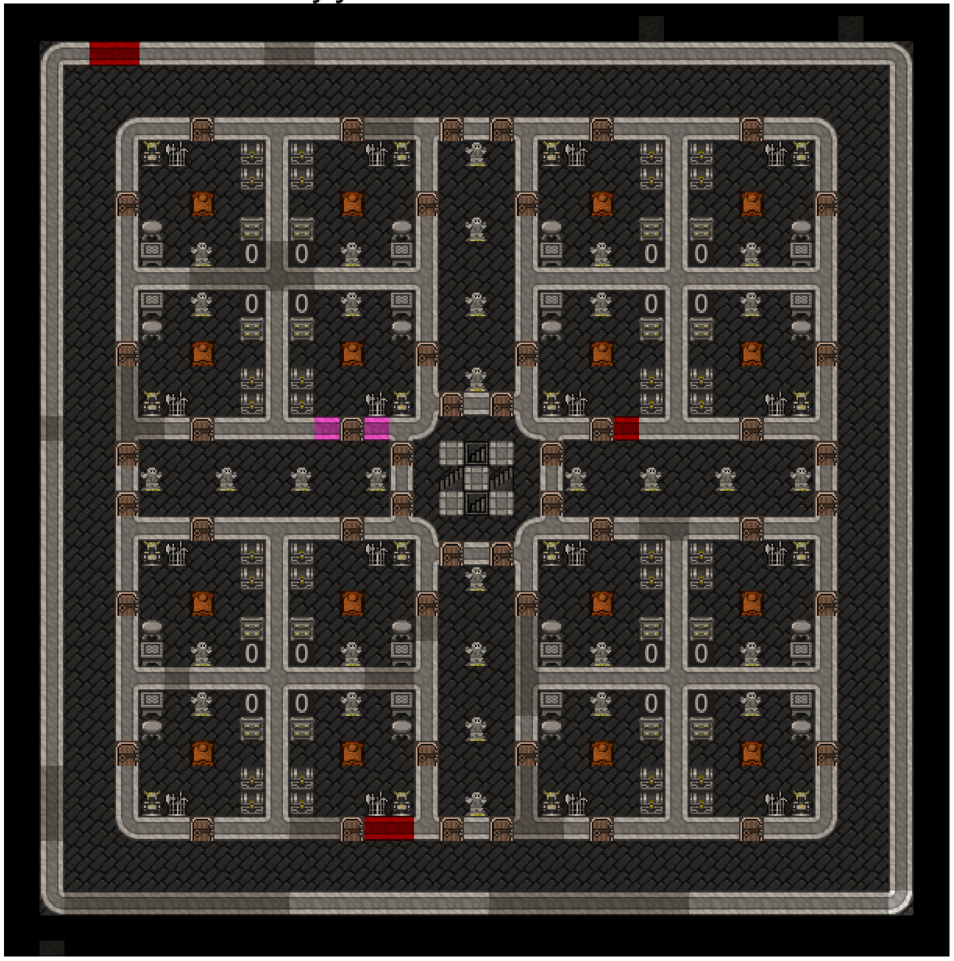
Guildhall level

Guildhalls, libraries, and temples, created as big empty rooms for you to customize or with optional prepared furniture layout (shown here)

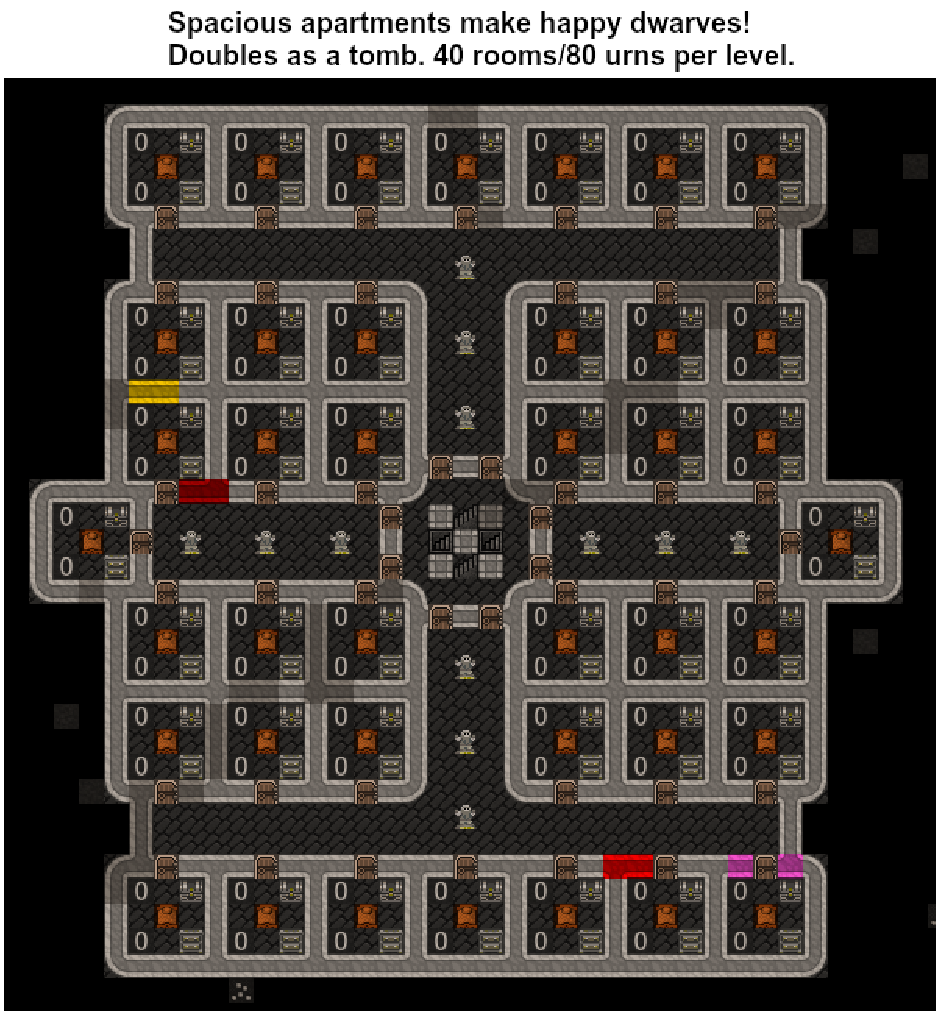


Noble suites

Designate and assign rooms as needed
to satisfy your nobles



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` or view the blueprints online.

5.3.2 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

5.3.3 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

5.3.4 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

5.3.5 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

5.3.6 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 information 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*.

The blueprint spreadsheet is 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 more convenient locations in your fort. See the step-by-step guide for using it by running `quickfort run library/pump_stack.csv -n /help`.

The blueprint spreadsheet is also available online.

5.4 Quickfort blueprint creation 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 *gui/quickfort* to browse and 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

5.4.1 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

5.4.2 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 #
Cw Cw    Cw #
#  #  #  #  #
```

Note my generosity – in addition to the bed (b) I've built a container (h) 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 sequences directly to your blueprints. For more details, check out the *Quickfort keystroke alias reference*. 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 #
d d d #
d d d #
# # # #
```

(continues on next page)

(continued from previous page)

```
#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
ga ga ga ga #
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
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 --
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 horizontal 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) ` #
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 especially 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
...
#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
#>
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 walkthrough"
```

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 reference* 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.

5.4.3 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 #
        #
        #
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.

5.4.4 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.

5.4.5 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.

5.4.6 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!

5.4.7 Caveats and limitations

- If you use the the `bags` alias, be aware that the game does not differentiate between empty and full bags. Therefore, you can 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.

5.4.8 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.

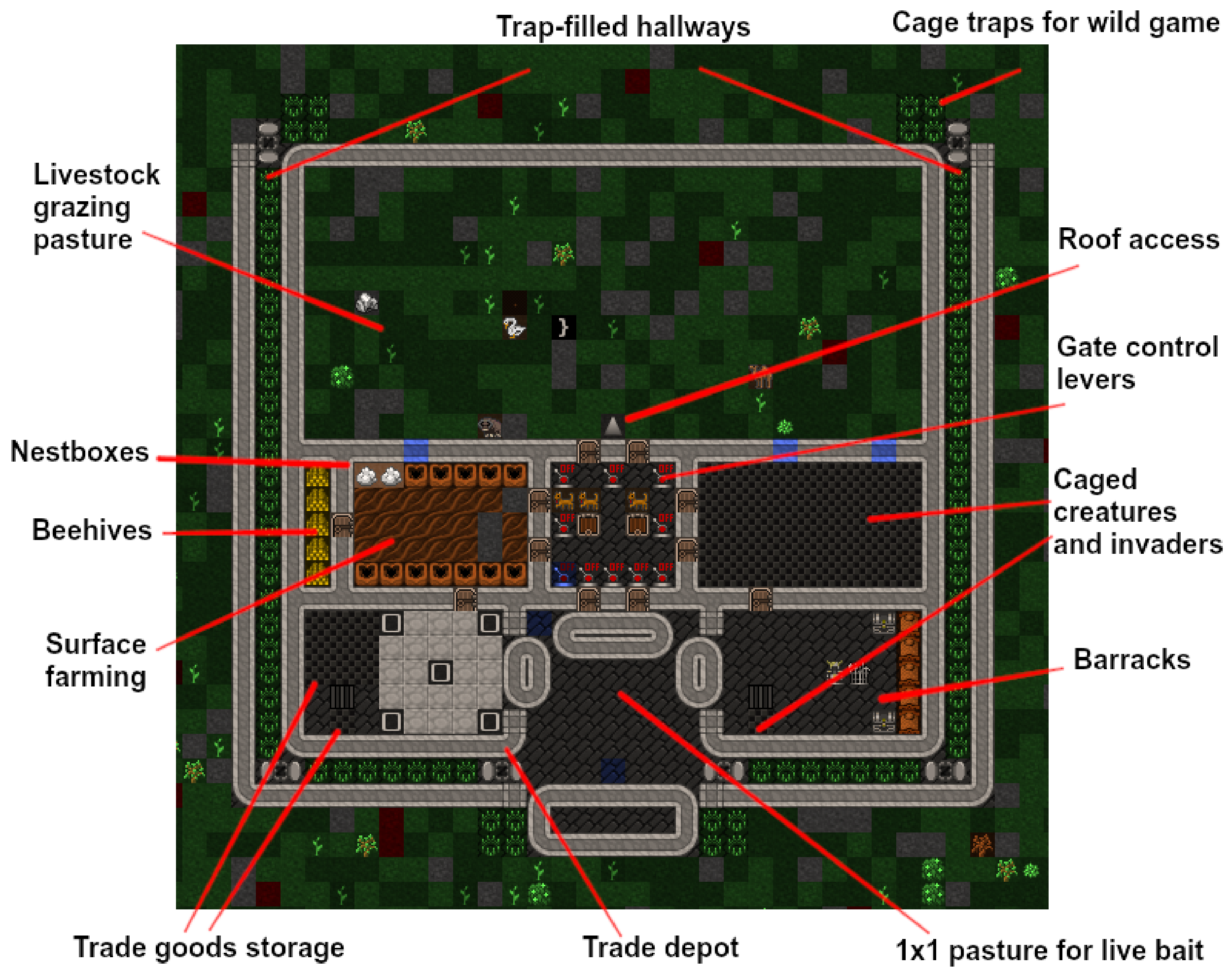
Do not neglect writing the help text! Not only will it give others a chance to use your blueprints appropriately, but the help you write will remind *you* what you were thinking when you wrote the blueprint in the first place.

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

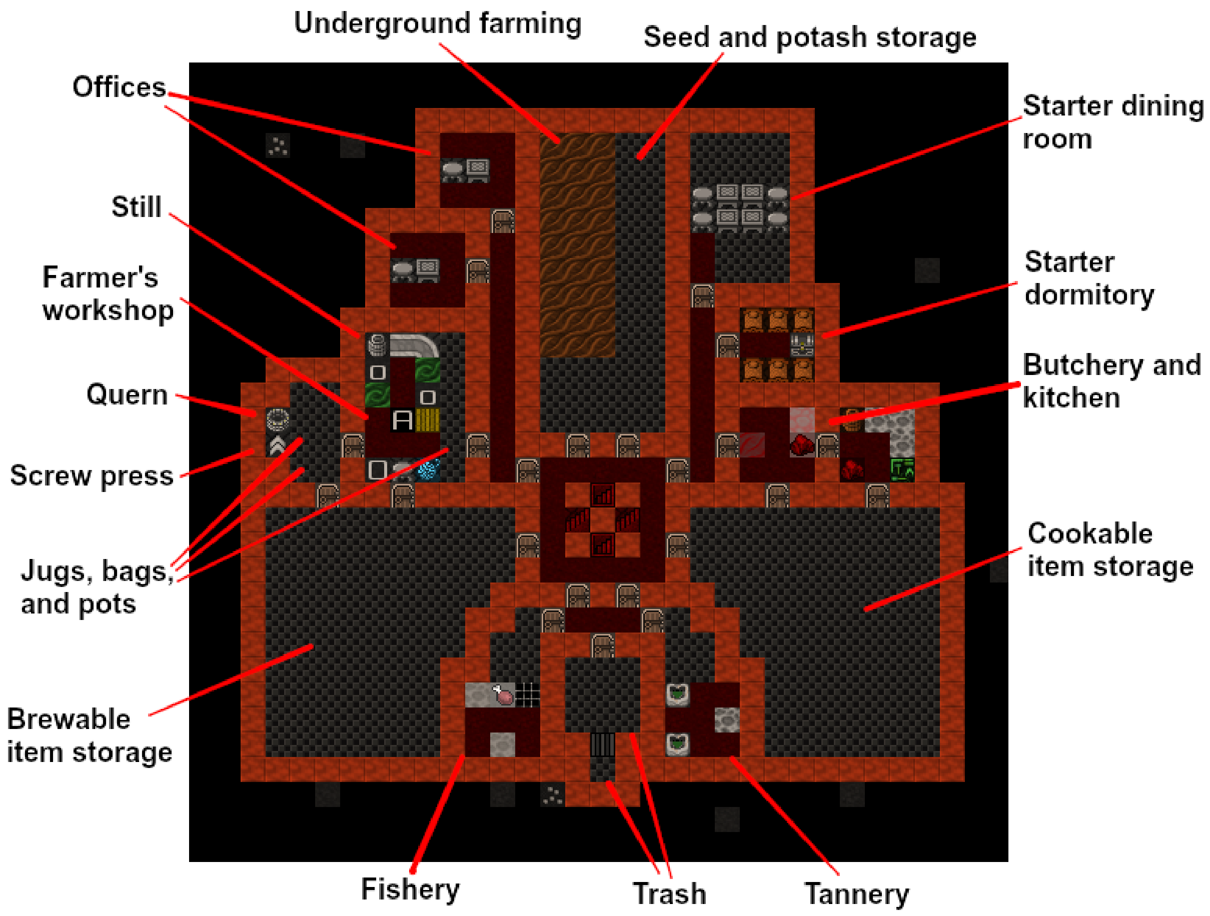
Use `message()` markers to remind yourself what to do next.

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.

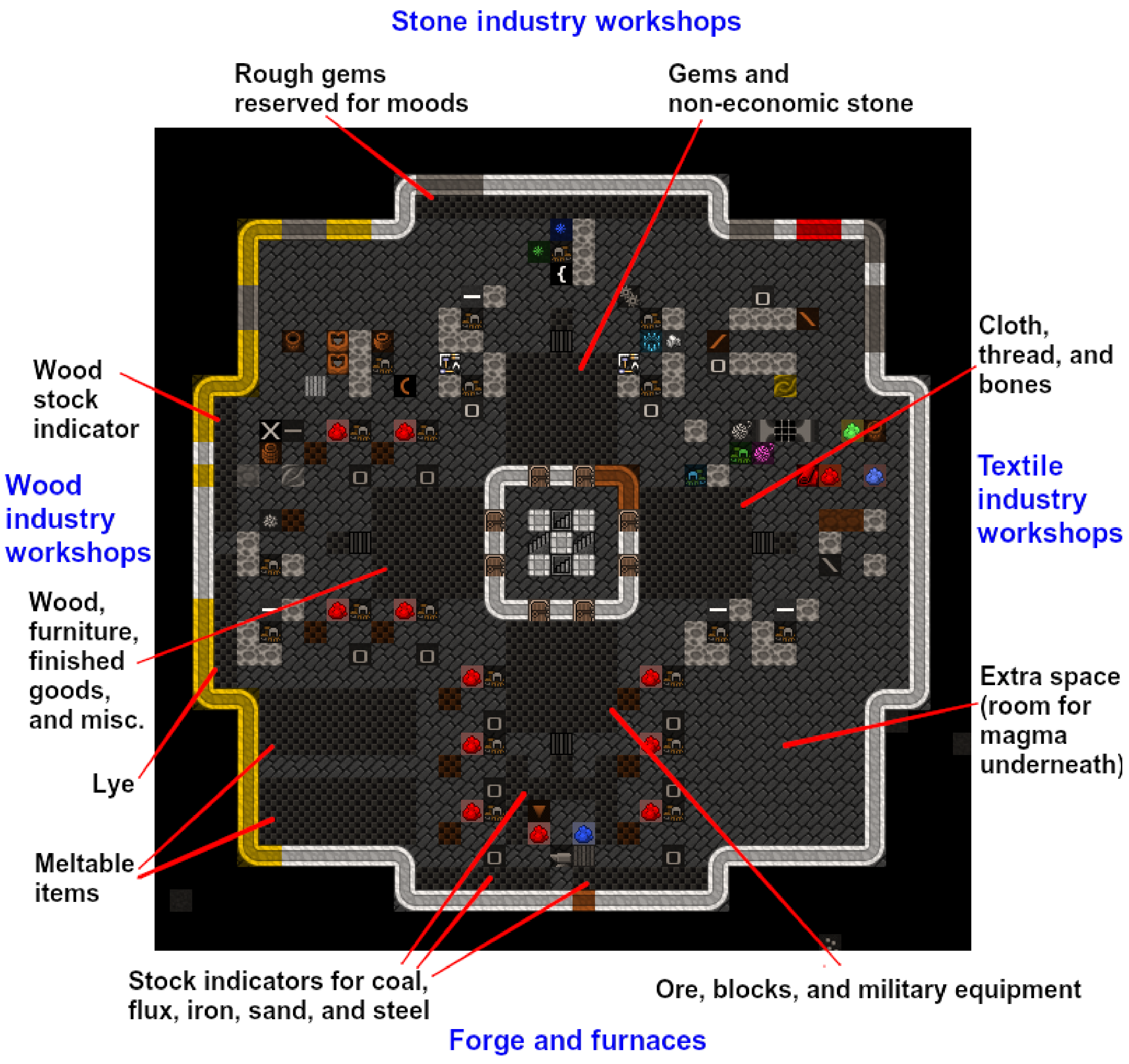
The farming level starts doing interesting things with `#query` blueprints and stockpiles. Note the careful customization of the food stockpiles and the stockpile chains set up with the `give*` aliases. This is so when multiple stockpiles can hold the same item, the largest can keep the smaller ones 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


Stone industry workshops
Rough gems reserved for moods
Gems and non-economic stone
Wood stock indicator
Wood industry workshops
Wood, furniture, finished goods, and misc.
Lye
Meltable items
Cloth, thread, and bones
Textile industry workshops
Extra space (room for magma underneath)
Stock indicators for coal, flux, iron, sand, and steel
Ore, blocks, and military equipment
Forge and furnaces

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:

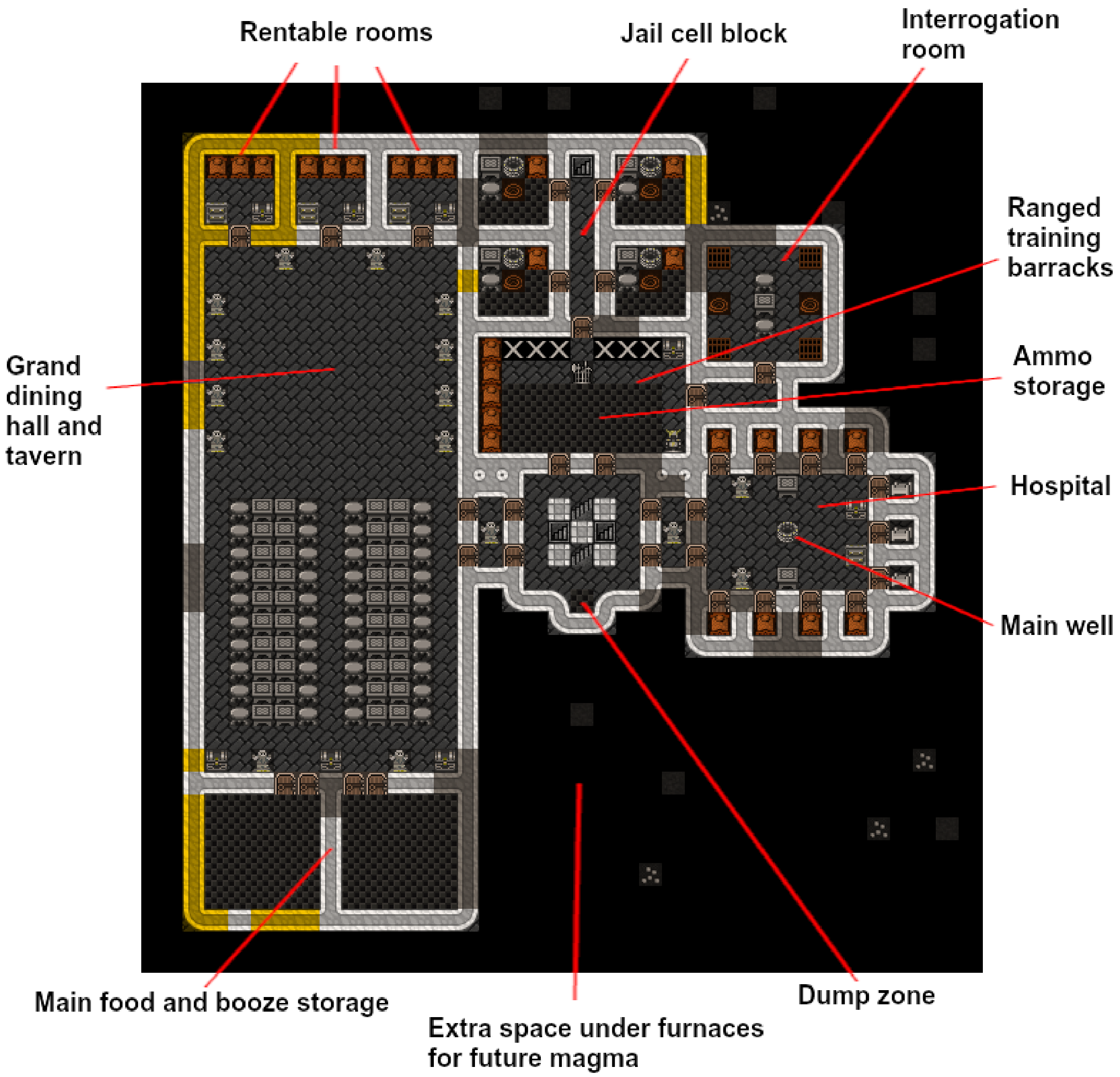
```
#place Declare a food stockpile
f(3x3)
#query Incorrectly configure a food stockpile to accept tallow and dye
tallow
permitdye
```

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*.

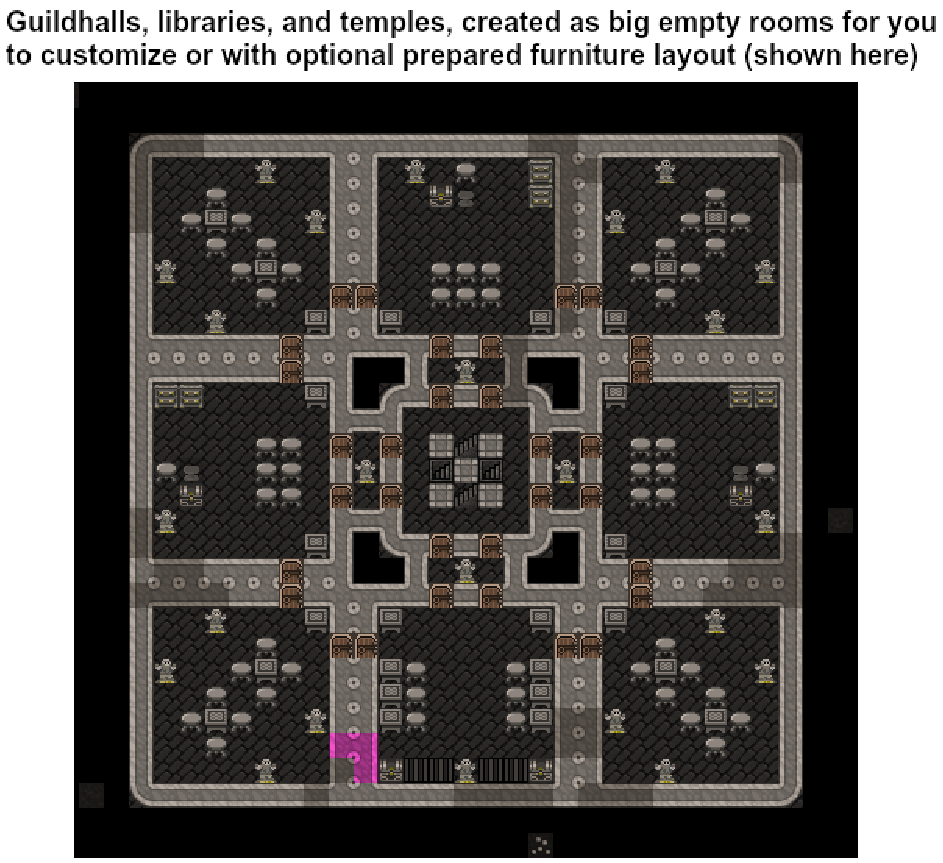
Tip

Use dig priorities to control ramp creation.

We can ensure the bottom level is carved out before the layer above is channeled by assigning the channel designations lower priorities (the `h5`s 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


Guildhalls, libraries, and temples, created as big empty rooms for you to customize or with optional prepared furniture layout (shown here)

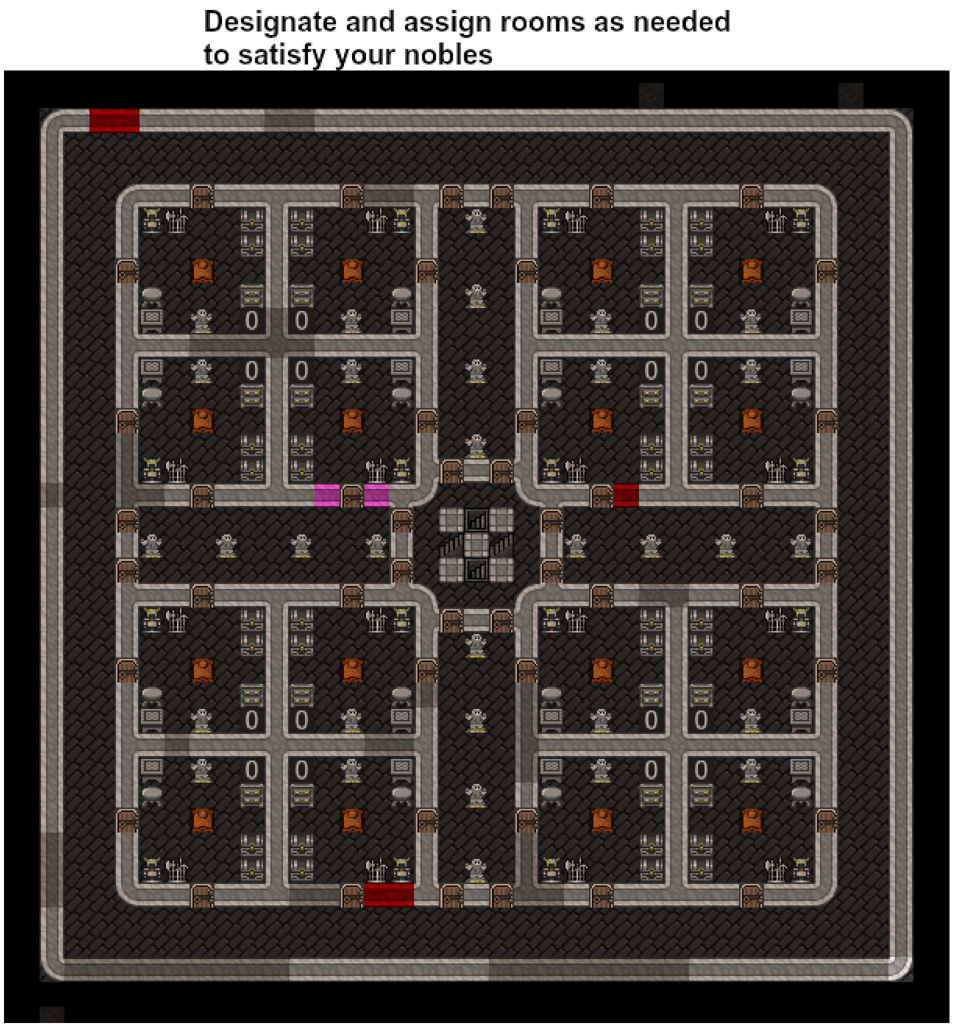
The goal of this level is to provide rooms for `locations` like guildhalls, libraries, and temples. The value of these rooms is very important, so we are likely to smooth and engrave everything. To smooth or engrave a wall tile, a dwarf has to be adjacent to it, and since some furniture, like statues, block dwarves from entering a tile, where you put them affects what you can access.

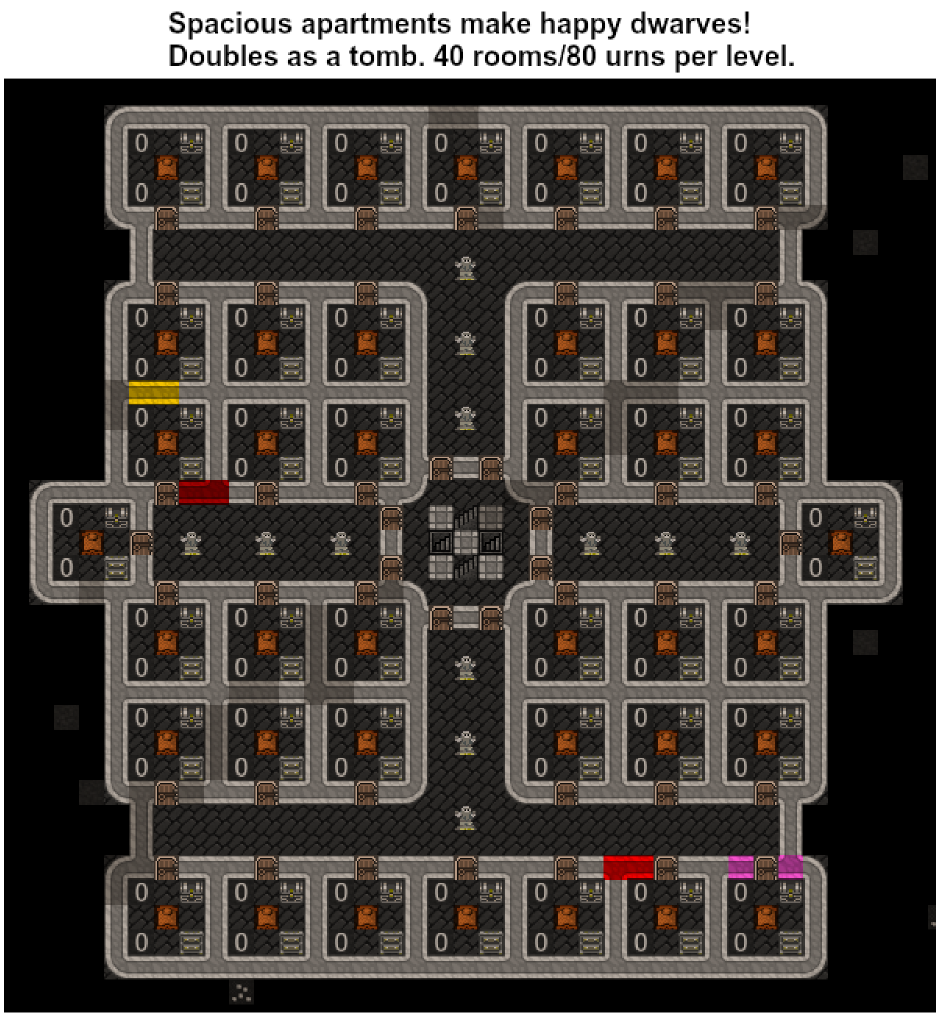
Tip

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


Designate and assign rooms as needed
to satisfy your nobles


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!

5.4.9 Links

Quickfort links:

- *Quickfort command reference*
- *Quickfort keystroke alias reference*
- *Quickfort blueprint library*
- 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.

5.5 Quickfort keystroke alias reference

Aliases allow you to use simple words to represent complicated key sequences when configuring buildings and stockpiles in quickfort `#query` and `#config` 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.

5.5.1 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.

5.5.2 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 necessary. 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}
```

5.5.3 Beyond query mode

`#query` blueprints normally do things in DF query 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 `query` 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 query mode.

5.5.4 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 configuration 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	
forbidsearch	search
permitsearch	search
togglesearch	search
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}^
```

`forbidsearch`, `permitsearch`, and `togglesearch` use the DFHack *search* plugin to forbid or permit a filtered list, or toggle the first (or only) item in the list. Specify the search string in the `search` sub-alias. Be sure to move the cursor over to the right column before invoking these aliases. The search filter will be cleared before this alias completes.

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 everything 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
usablehair	forbidusablehair	permitusablehair
craftrefuse	forbidcraftrefuse	permitcraftrefuse

Notes:

- `usablehair` Only hair and wool that can make usable clothing is included, i.e. from sheep, llamas, alpacas, and trolls.
- `craftrefuse` includes everything a craftsdwarf or tailor can use: skulls, bones, shells, teeth, horns, and "usable" hair/wool (defined above).

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
stonetools		
woodentools		
crafts	forbidcrafts	permitcrafts
goblets	forbidgoblets	permitgoblets
masterworkfinishedgoods	forbidmasterworkfinishedgoods	permitmasterworkfinishedgoods
artifactfinishedgoods	forbidartifactfinishedgoods	permitartifactfinishedgoods

Cloth stockpile adjustments

Exclusive	Forbid	Permit
thread	forbidthread	permitthread
adamantinethread	forbidadamantinethread	permitadamantinethread
cloth	forbidcloth	permitcloth
adamantinecloth	forbidadamantinecloth	permitadamantinecloth

Notes:

- `thread` and `cloth` refers to all materials that are not adamantine.

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

CHAPTER

SIX

DFHACK DEVELOPMENT GUIDE

These are pages relevant to people developing for DFHack.

6.1 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

6.1.1 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/examples` 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 *disable* commands. For the full plugin API, see the example `skeleton` plugin 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.

Run *plug* at the DFHack prompt for a list of all plugins included in DFHack.

6.1.2 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.

6.1.3 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.

6.1.4 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++.

6.1.5 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.

6.2 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

6.2.1 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*.

6.2.2 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.

6.2.3 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 compatibility 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.

6.2.4 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 defining 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.

6.2.5 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 download 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 Manager. 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`.

6.2.6 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.

6.2.7 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.

6.3 How to contribute to DFHack

Contents

6.3.1 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 help* 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 `docs/changelog.txt` and `docs/about/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.

6.3.2 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!

6.4 DFHack documentation system

DFHack documentation, like the file you are reading now, is created as a set of `.rst` files 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 named Sphinx.

The DFHack build process will compile and install the documentation so it can be displayed in-game by the *help* and *ls* commands (and any other command or GUI that displays help text), but documentation compilation is disabled by default due to the additional Python and Sphinx requirements. If you already have a version of the docs installed (say from a downloaded release binary), then you only need to build the docs if you're changing them and want to see the changes reflected in your game.

You can also build the docs if you just want a local HTML- or text-rendered copy, though you can always read the online version too. The active development version of the documentation is tagged with `latest` and is available here

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

6.4.1 Concepts and general guidance

The source `.rst` files are compiled to HTML for viewing in a browser and to text format for viewing in-game. For in-game help, the help text is read from its installed location in `hack/docs` under the DF directory.

When writing documentation, remember that everything should be documented! If it's not clear *where* a particular thing should be documented, ask on Discord or in the DFHack thread on Bay12 – you'll not only be getting help, you'll also be providing valuable feedback that makes it easier for future contributors to find documentation on how to write the documentation!

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.

Short descriptions

Each command that a user can run – as well as every plugin – needs to have a short (~54 character) descriptive string associated with it. This description text is:

- used in-game by the *ls* command and DFHack UI screens that list commands
- used in the generated index entries in the HTML docs

Tags

To make it easier for players to find related commands, all plugins and commands are marked with relevant tags. These are used to compile indices and generate cross-links between the commands, both in the HTML documents and in-game. See the list of available tag-list and think about which categories your new tool belongs in.

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*. Scripts and plugins have link targets that match their names created for you automatically.

If you want to link to a heading in your own page, you can specify it like this:

```
`Heading text exactly as written`_
```

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 outside of DFHack.

6.4.2 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.

Where do I add the help text?

For scripts and plugins that are distributed as part of DFHack, documentation files should be added to the scripts/docs and docs/plugins directories, respectively, in a file named after the script or plugin. For example, a script named `gui/foobar.lua` (which provides the `gui/foobar` command) should be documented in a file named `docs/gui/foobar.rst` in the scripts repo. Similarly, a plugin named `foobaz` should be documented in a file named `docs/plugins/foobaz.rst` in the dfhack repo. For plugins, all commands provided by that plugin should be documented in that same file.

Short descriptions (the ~54 character short help) for scripts and plugins are taken from the `summary` attribute of the `dfhack-tool` directive that each tool help document must have (see the *Header format* section below). Please make this brief but descriptive!

Short descriptions for commands provided by plugins are taken from the `description` parameter passed to the `PluginCommand` constructor used when the command is registered in the plugin source file.

Header format

The docs **must** begin with a heading which exactly matches the script or plugin name, underlined with ===== to the same length. This must be followed by a `.. dfhack-tool:` directive with at least the following parameters:

- `:summary:` - a short, single-sentence description of the tool
- `:tags:` - a space-separated list of tags that apply to the tool

By default, `dfhack-tool` generates both a description of a tool and a command with the same name. For tools (specifically plugins) that do not provide exactly 1 command with the same name as the tool, pass the `:no-command:` parameter (with no content after it) to prevent the command block from being generated.

For tools that provide multiple commands, or a command by the same name but with significantly different functionality (e.g. a plugin that can be both enabled and invoked as a command for different results), use the `.. dfhack-command:` directive for each command. This takes only a `:summary:` argument, with the same meaning as above.

For example, documentation for the `build-now` script might look like:

```
build-now
=========

.. dfhack-tool::
    :summary: Instantly completes unsuspended building construction jobs.
    :tags: fort armok buildings

By default, all buildings on the map are completed, but the area of effect is␣
↪configurable.
```

And documentation for the `autodump` plugin might look like:

```
autodump
========

```

(continues on next page)

(continued from previous page)

```
.. dfhack-tool::
    :summary: Automatically set items in a stockpile to be dumped.
    :tags: fort armok fps productivity items stockpiles
    :no-command:

.. dfhack-command:: autodump
    :summary: Teleports items marked for dumping to the cursor position.

.. dfhack-command:: autodump-destroy-here
    :summary: Destroy items marked for dumping under the cursor.

.. dfhack-command:: autodump-destroy-item
    :summary: Destroys the selected item.

When `enabled <enable>`, this plugin adds an option to the :kbd:`q` menu for
stockpiles.

When invoked as a command, it can instantly move all unforbidden items marked
for dumping to the tile under the cursor.
```

Usage help

The first section after the header and introductory text should be the usage section. You can choose between two formats, based on whatever is cleaner or clearer for your syntax. The first option is to show usage formats together, with an explanation following the block:

```
Usage
-----

::

    build-now [<options>]
    build-now here [<options>]
    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.
```

The second option is to arrange the usage options in a list, with the full command and arguments in monospaced font. Then indent the next line and describe the effect:

```
Usage
-----
```

(continues on next page)

(continued from previous page)

```
``build-now [<options>]``
    Scan the entire map and build all unsuspended constructions
    and buildings.
``build-now here [<options>]``
    Build the unsuspended construction or building under the
    cursor.
``build-now [<pos> [<pos>]] [<options>]``
    Build all unsuspended constructions within the specified
    coordinate box.

The ``<pos>`` parameters are specified as...
```

Note that in both options, the entire commandline syntax is written, including the command itself. Literal text is written as-is (e.g. the word `here` in the above example), and text that describes the kind of parameter that is being passed (e.g. `pos` or `options`) is enclosed in angle brackets (`<` and `>`). Optional elements are enclosed in square brackets (`[` and `]`). If the command takes an arbitrary number of elements, use `...`, for example:

```
prioritize [<options>] <job type> [<job type> ...]
quickfort <command>[,<command>...] <list_id>[,<list_id>...] [<options>]
```

Examples

If the only way to run the command is to type the command itself, then this section is not necessary. Otherwise, please consider adding a section that shows some real, practical usage examples. For many users, this will be the **only** section they will read. It is so important that it is a good idea to include the `Examples` section **before** you describe any extended options your command might take. Write examples for what you expect the popular use cases will be. Also be sure to write examples showing specific, practical values being used for any parameter that takes a value or has tricky formatting.

Examples should go in their own subheading. The examples themselves should be organized as in option 2 for Usage above. Here is an example `Examples` section:

```
Examples
--------

``build-now``
    Completes all unsuspended construction jobs on the map.
``build-now 37,20,154 here``
    Builds the unsuspended, unconstructed buildings in the box
    bounded by the coordinate x=37,y=20,z=154 and the cursor.
```

Options

The options header should follow the examples, with each option in the same format as the examples:

```
Options
-------

``-h``, ``--help``
    Show help text.
``-l``, ``--quality <level>``
```

(continues on next page)

(continued from previous page)

```
    Set the quality of the architecture for built architected
    builtings.
``-q``, ``--quiet``
    Suppress informational output (error messages are still
    printed).
```

Note that for parameters that have both short and long forms, any values that those options take only need to be specified once (e.g. `<level>`).

6.4.3 External scripts and plugins

Scripts and plugins distributed separately from DFHack's release packages don't have the opportunity to add their documentation to the rendered HTML or text output. However, these scripts and plugins can use a different mechanism to at least make their help text available in-game.

Note that since help text for external scripts and plugins is not rendered by Sphinx, it should be written in plain text. Any reStructuredText markup will not be processed and, if present, will be shown verbatim to the player (which is probably not what you want).

For external scripts, the short description comes from a comment on the first line (the comment marker and extra whitespace is stripped). For Lua, this would look like:

```
-- A short description of my cool script.
```

and for Ruby scripts it would look like:

```
# A short description of my cool script.
```

The main help text for an external script needs to appear between two markers. For Lua, these markers are `[====[` and `]====]`, and for Ruby they are `=begin` and `=end`. The documentation standards above still apply to external tools, but there is no need to include backticks for links or monospaced fonts. Here is a Lua example for an entire script header:

```
-- Inventory management for adventurers.
-- [====[
gui/adv-inventory
=================

Tags: adventure | items

Allows you to quickly move items between containers. This
includes yourself and any followers you have.

Usage
-----

    gui/adv-inventory [<options>]

Examples
--------

gui/adv-inventory
    Opens the GUI with nothing preselected

```

(continues on next page)

(continued from previous page)

```
gui/adv-inventory take-all
    Opens the GUI with all container items already selected and
    ready to move into the adventurer's inventory.

Options
-------

take-all
    Starts the GUI with container items pre-selected

give-all
    Starts the GUI with your own items pre-selected
]====]
```

For external plugins, help text for provided commands can be passed as the `usage` parameter when registering the commands with the `PluginCommand` constructor. There is currently no way for associating help text with the plugin itself, so any information about what the plugin does when enabled should be combined into the command help.

6.4.4 Required dependencies

In order to build the documentation, you must have Python with Sphinx version 3.4.3 or later and Python 3.

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 3. On Debian-based distros:

```
sudo apt install python3
```

Check your package manager to see if Sphinx 3.4.3 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
```

Windows

Python for Windows can be downloaded from python.org. The latest version of Python 3 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.

6.4.5 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 the batch script to add the flag. You can also run `cmake` on the command line, similar to other platforms.

By default, both HTML and text docs are built by CMake. The generated documentation is stored in `docs/html` and `docs/text` (respectively) in the root DFHack folder, and they will both be installed to `hack/docs` when you install DFHack. The html and txt files will intermingle, but will not interfere with one another.

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. The `docs/build.py` script will build the documentation in any specified formats (HTML only by default) using the same command that CMake runs when building the docs. Run the script with `--help` to see additional options.

Examples:

- **`docs/build.py`** Build just the HTML docs
- **`docs/build.py html text`** Build both the HTML and text docs
- **`docs/build.py --clean`** Build HTML and force a clean build (all source files are re-read)

The resulting documentation will be stored in `docs/html` and/or `docs/text`.

Alternatively, you can run Sphinx manually with:

```
sphinx-build . docs/html
```

or, to build plain-text output:

```
sphinx-build -b text . docs/text
```

Sphinx has many options to enable clean builds, parallel builds, logging, and more - run `sphinx-build --help` for details. If you specify a different output path, be warned that Sphinx may overwrite existing files in the output folder. Also be aware that when running `sphinx-build` directly, the `docs/html` folder may be polluted with intermediate build files that normally get written in the cmake `build` directory.

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:

```
docs/build.py pdf
```

Alternatively, you can run Sphinx manually with:

```
sphinx-build -M latexpdf . docs/pdf
```

6.4.6 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/changelogs/news.rst` and `docs/changelogs/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.

6.4.7 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.

6.5 DFHack API concepts

6.5.1 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".

6.6 DFHack Lua API Reference

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

6.6.1 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.

6.6.2 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.getCommandHistory(history_id, history_filename)`

  Returns the list of strings in the specified history. Intended to be used by GUI scripts that don't have access to a console and so can't use `dfhack.lineedit`. The `history_id` parameter is some unique string that the script uses to identify its command history, such as the script's name. If this is the first time the history with the given `history_id` is being accessed, it is initialized from the given file.

- `dfhack.addCommandToHistory(history_id, history_filename, command)`

  Adds a command to the specified history and saves the updated history to the specified file.

- `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, returns *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

- *Internal API*

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()`
- `dfhack.isPrerelease()`

  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', 'quick'})
dfhack.run_command('ls', 'quick')
dfhack.run_command('ls quick')  -- 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.pauseRecenter(pos[,pause])` `dfhack.gui.pauseRecenter(x,y,z[,pause])`

  Same as `resetDwarfmodeView`, but also recenter if position is valid. If `pause` is false, skip pausing. Respects `RECENTER_INTERFACE_SHUTDOWN_MS` in DF's `init.txt` (the delay before input is recognized when a recenter occurs.)

- `dfhack.gui.revealInDwarfmodeMap(pos[,center])` `dfhack.gui.revealInDwarfmodeMap(x,y,z[, center])`

  Centers the view on the given coordinates. If `center` is true, make sure the position is in the exact center of the view, else just bring it on screen.

  `pos` 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)
  ```

  If the position is invalid, the function will simply ensure the current window position is clamped between valid values.

- `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`.

- `dfhack.gui.autoDFAnnouncement(report,text)` `dfhack.gui.autoDFAnnouncement(type,pos,text,color[,is_bright,unit1,unit2,is_sparring])`

  Takes a `df.report_init` (see: structure definition) and a string and processes them just like DF does. Can also be built from parameters instead of a `report_init`. Setting `is_sparring` to *true* means the report will be added to sparring logs (if applicable) rather than hunting or combat.

  The announcement will not display if units are involved and the player can't see them (or hear, for adventure mode sound announcement types.) Text is parsed using `&` as an escape character, with `&r` adding a blank line (equivalent to `\n \n`,) `&&` being just `&`, and any other combination causing neither character to display.

  If you want a guaranteed announcement without parsing, use `dfhack.gui.showAutoAnnouncement` instead.

- `dfhack.gui.getMousePos()`

  Returns the map coordinates of the map tile the mouse is over as a table of `{x, y, z}`. If the cursor is not over the map, returns `nil`.

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 overridden 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.isUnitInBox(unit,x1,y1,z1,x2,y2,z2)`

  The unit is within the specified coordinates.

- `dfhack.units.isActive(unit)`

  The unit is active (alive and on the map).

- `dfhack.units.isVisible(unit)`

  The unit is visible on the map.

- `dfhack.units.isCitizen(unit[,ignore_sanity])`

  The unit is an alive sane citizen of the fortress; wraps the same checks the game uses to decide game-over by extinction, with an additional sanity check. You can identify citizens, regardless of their sanity, by passing `true` as the optional second parameter.

- `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.isOwnCiv(unit)`

  The unit belongs to the player's civilization.

- `dfhack.units.isOwnGroup(unit)`

  The unit belongs to the player's group.

- `dfhack.units.isOwnRace(unit)`

  The unit belongs to the player's race.

- `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.isSane(unit)`

  The unit is capable of rational action, i.e. not dead, insane, zombie, or active werewolf.

- `dfhack.units.isCrazed`

  The unit is berserk and will attack all other creatures except members of its own species that are also crazed. (can be modified by curses)

- `dfhack.units.isGhost(unit)`

  The unit is a ghost.

- `dfhack.units.isHidden(unit)`

  The unit is hidden to the player, accounting for sneaking. Works for any game mode.

- `dfhack.units.isHidingCurse(unit)`

  The unit is hiding a curse.

- `dfhack.units.isMale(unit)`
- `dfhack.units.isFemale(unit)`
- `dfhack.units.isBaby(unit)`
- `dfhack.units.isChild(unit)`
- `dfhack.units.isAdult(unit)`
- `dfhack.units.isGay(unit)`
- `dfhack.units.isNaked(unit)`

  Simple unit property checks

- `dfhack.units.isVisiting(unit)`

  The unit is visiting. eg. Merchants, Diplomatics, travelers.

- `dfhack.units.isTrainableHunting(unit)`

  The unit is trainable for hunting.
- `dfhack.units.isTrainableWar(unit)`

  The unit is trainable for war.
- `dfhack.units.isTrained(unit)`

  The unit is trained.
- `dfhack.units.isHunter(unit)`

  The unit is a trained hunter.
- `dfhack.units.isWar(unit)`

  The unit is trained for war.
- `dfhack.units.isTame(unit)`
- `dfhack.units.isTamable(unit)`
- `dfhack.units.isDomesticated(unit)`
- `dfhack.units.isMarkedForSlaughter(unit)`
- `dfhack.units.isGelded(unit)`
- `dfhack.units.isEggLayer(unit)`
- `dfhack.units.isGrazer(unit)`
- `dfhack.units.isMilkable(unit)`

  Simple unit property checks.
- `dfhack.units.isForest(unit)`

  The unit is of the forest.
- `dfhack.units.isMischievous(unit)`

  The unit is mischievous.
- `dfhack.units.isAvailableForAdoption(unit)`

  The unit is available for adoption.
- `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.isDwarf(unit)`

  The unit is of the correct race for the fortress.
- `dfhack.units.isAnimal(unit)`
- `dfhack.units.isMerchant(unit)`
- `dfhack.units.isDiplomat(unit)`

  Simple unit type checks.

- `dfhack.units.isVisitor(unit)`

  The unit is a regular visitor with no special purpose (eg. merchant).

- `dfhack.units.isInvader(unit)`

  The unit is an active invader or marauder.

- `dfhack.units.isUndead(unit[,include_vamps])`

  The unit is undead. Pass `true` as the optional second parameter to count vampires as undead.

- `dfhack.units.isNightCreature(unit)`
- `dfhack.units.isSemiMegabeast(unit)`
- `dfhack.units.isMegabeast(unit)`
- `dfhack.units.isTitan(unit)`
- `dfhack.units.isDemon(unit)`

  Simple enemy type checks.

- `dfhack.units.isDanger(unit)`

  The unit is dangerous, and probably hostile. This includes Great Dangers (see below), semi-megabeasts, night creatures, undead, invaders, and crazed units.

- `dfhack.units.isGreatDanger(unit)`

  The unit is of Great Danger. This include demons, titans, and megabeasts.

- `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.getPhysicalAttrValue(unit, attr_type)`
- `dfhack.units.getMentalAttrValue(unit, attr_type)`

  Computes the effective attribute value, including curse effect.

- `dfhack.units.getMiscTrait(unit, type[, create])`

  Finds (or creates if requested) a misc trait object with the given id.

- `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.

Action Timer API

This is an API to allow manipulation of unit action timers, to speed them up or slow them down. All functions in this API have overflow/underflow protection when modifying action timers (the value will cap out). Actions with a timer of 0 (or less) will not be modified as they are completed (or invalid in the case of negatives). Timers will be capped to go no lower than 1. `affectedActionType` parameters are integers from the DF enum `unit_action_type`. E.g. `df.unit_action_type.Move`. `affectedActionTypeGroup` parameters are integers from the (custom) DF enum `unit_action_type_group`. They are as follows:

- `All` (does not include unknown unit action types)
- `Movement`
- `MovementFeet` (affects only walking and crawling speed. if you need to differentiate between walking and crawling, check the unit's `flags1.on_ground` flag, like the Pegasus boots do in the *DFHack modding guide*)
- `MovementFeet` (for walking speed, such as with pegasus boots from the *DFHack modding guide*)
- `Combat` (includes bloodsucking)
- `Work`

API functions:

- `subtractActionTimers(unit, amount, affectedActionType)`

  Subtract `amount` (32-bit integer) from the timers of any actions the unit is performing of `affectedActionType` (usually one or zero actions in normal gameplay).

- `subtractGroupActionTimers(unit, amount, affectedActionTypeGroup)`

  Subtract `amount` (32-bit integer) from the timers of any actions the unit is performing that match the `affectedActionTypeGroup` category.

- `multiplyActionTimers(unit, amount, affectedActionType)`

  Multiply the timers of any actions of `affectedActionType` the unit is performing by `amount` (float) (usually one or zero actions in normal gameplay).

- `multiplyGroupActionTimers(unit, amount, affectedActionTypeGroup)`

  Multiply the timers of any actions that match the `affectedActionTypeGroup` category the unit is performing by `amount` (float).

- `setActionTimers(unit, amount, affectedActionType)`

  Set the timers of any action the unit is performing of `affectedActionType` to `amount` (32-bit integer) (usually one or zero actions in normal gameplay).

- `setGroupActionTimers(unit, amount, affectedActionTypeGroup)`

  Set the timers of any action the unit is performing that match the `affectedActionTypeGroup` category to `amount` (32-bit integer).

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 accommodate 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.

- `dfhack.constructions.findAtTile(pos)`, or `findAtTile(x,y,z)`

  Returns the construction at the given position, or `nil` if there isn't one.

- `dfhack.constructions.insert(construction)`

  Properly inserts the given construction into the game. Returns false and fails to insert if there was already a construction at the position.

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.hideGuard(screen,callback[,args...])`

  Removes screen from the viewscreen stack, calls the callback (with optional supplied arguments), and then restores the screen on the top of the viewscreen stack.

- `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 keybindings 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 canceled 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`.

6.6.3 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 extensions

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 delimiter 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 hyphens (`-`) 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 argruments 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).

- `argparse.positiveInt(arg, arg_name)`

  Throws if `tonumber(arg)` is not a positive integer; otherwise returns `tonumber(arg)`. If `arg_name` is specified, it is used to make error messages more useful.

- `argparse.nonnegativeInt(arg, arg_name)`

  Throws if `tonumber(arg)` is not a non-negative integer; otherwise returns `tonumber(arg)`. If `arg_name` is specified, it is used to make error messages more useful.

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.

helpdb

Unified interface for DFHack tool help text. Help text is read from the rendered text in `hack/docs/docs/`. If no rendered text exists, help is read from the script sources (for scripts) or the string passed to the `PluginCommand` initializer (for plugins). See *DFHack documentation system* for details on how DFHack's help system works.

The database is lazy-loaded when an API method is called. It rechecks its help sources for updates if an API method has not been called in the last 60 seconds.

Each entry has several properties associated with it:

- The entry name, which is the name of a plugin, script, or command provided by a plugin.
- The entry types, which can be `builtin`, `plugin`, and/or `command`. Entries for built-in commands (like `ls` or `quicksave`) are both type `builtin` and `command`. Entries named after plugins are type `plugin`, and if that plugin also provides a command with the same name as the plugin, then the entry is also type `command`. Entry types are returned as a map of one or more of the type strings to `true`.
- Short help, a the ~54 character description string.
- Long help, the entire contents of the associated help file.
- A list of tags that define the groups that the entry belongs to.
- `helpdb.is_entry(str)`, `helpdb.is_entry(list)`

  Returns whether the given string (or list of strings) is an entry (are all entries) in the db.
- `helpdb.get_entry_types(entry)`

  Returns the set (that is, a map of string to `true`) of entry types for the given entry.
- `helpdb.get_entry_short_help(entry)`

  Returns the short (~54 character) description for the given entry.
- `helpdb.get_entry_long_help(entry[, width])`

  Returns the full help text for the given entry. If `width` is specified, the text will be wrapped at that width, preserving block indents. The wrap width defaults to 80.
- `helpdb.get_entry_tags(entry)`

  Returns the set of tag names for the given entry.
- `helpdb.is_tag(str)`, `helpdb.is_tag(list)`

  Returns whether the given string (or list of strings) is a (are all) valid tag name(s).

- `helpdb.get_tags()`

  Returns the full alphabetized list of valid tag names.

- `helpdb.get_tag_data(tag)`

  Returns a list of entries that have the given tag. The returned table also has a `description` key that contains the string description of the tag.

- `helpdb.search_entries([include[, exclude]])`

  Returns a list of names for entries that match the given filters. The list is alphabetized by their last path component, with populated path components coming before null path components (e.g. `autobutcher` will immediately follow `gui/autobutcher`). The optional `include` and `exclude` filter params are maps (or lists of maps) with the following elements:

  str if a string, filters by the given substring. if a table of strings, includes entry names that match any of the given substrings.

  tag if a string, filters by the given tag name. if a table of strings, includes entries that match any of the given tags.

  entry_type if a string, matches entries of the given type. if a table of strings, includes entries that match any of the given types.

  Elements in a map are ANDed together (e.g. if both `str` and `tag` are specified, the match is on any of the `str` elements AND any of the `tag` elements).

  If lists of filters are passed instead of a single map, the maps are ORed (that is, the match succeeds if any of the filters match).

  If `include` is `nil` or empty, then all entries are included. If `exclude` is `nil` or empty, then no entries are filtered out.

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 a 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()

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 direct 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 the 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 applicable. For plant 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.

It is recommended to prefix custom raw tokens with the name of your mod to avoid duplicate behaviour where two mods make callbacks that work on the same tag.

Examples:

- Using an eventful onReactionComplete hook, something for disturbing dwarven science:

```lua
if customRawTokens.getToken(reaction, "EXAMPLE_MOD_CAUSES_INSANITY") then
    -- make unit who performed reaction go insane
```

- Using an eventful onProjItemCheckMovement hook, a fast or slow-firing crossbow:

```lua
-- check projectile distance flown is zero, get firer, etc...
local multiplier = tonumber(customRawTokens.getToken(bow, "EXAMPLE_MOD_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:

```lua
local unit = dfhack.gui.getSelectedUnit()
if not unit then return end
local helpText = customRawTokens.getToken(unit, "EXAMPLE_MOD_HELP_TEXT")
if helpText then print(helpText) end
```

- Healing armour:

```lua
-- (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,
↪"EXAMPLE_MOD_HEAL_AMOUNT")) or 0)
    end
```

(continues on next page)

(continued from previous page)

```
end
unit.body.blood_count = math.min(unit.body.blood_max, unit.body.blood_count +␣
↪healAmount)
```

6.6.4 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 use by script writers and should not be set by library widgets for their internal subviews.

on_focus Called when the view gains keyboard focus; see `setFocus()` below.

on_unfocus Called when the view loses keyboard focus.

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.

parent_view A reference to the parent view. This field is `nil` until the view is added as a subview to another view with `addviews()`.

focus_group The list of widgets in a hierarchy. This table is unique and empty when a view is initialized, but is replaced by a shared table when the view is added to a parent via `addviews()`. If a view in the focus group has keyboard focus, that widget can be accessed via `focus_group.cur`.

focus A boolean indicating whether the view currently has keyboard focus.

These fields are computed by the layout process:

frame_parent_rect The ViewRect representing 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([view_rect])`

  Returns the mouse *x,y* in coordinates local to the given ViewRect (or `frame_body` if no ViewRect is passed) 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. If a subview within the view's `focus_group` has focus and it and all of its ancestors are active and visible, that subview is offered the chance to handle the input before any other subviews.

- `view:getPreferredFocusState()`

  Returns `false` by default, but should be overridden by subclasses that may want to take keyboard focus (if it is unclaimed) when they are added to a parent view with `addviews()`.

- `view:setFocus(focus)`

  Sets the keyboard focus to the view if `focus` is `true`, or relinquishes keyboard focus if `focus` is `false`. Views that newly acquire keyboard focus will trigger the `on_focus` callback, and views that lose keyboard focus will trigger the `on_unfocus` callback. While a view has focus, all keyboard input is sent to that view before any of its siblings or parents. Keyboard input is propagated as normal (see `inputToSubviews()` above) if there is no view with focus or if the view with focus returns `false` from its `onInput()` function.

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. This function returns `self` as a convenience so you can write such code as `local view = MyScreen{params=val}:show()`.

- `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 height 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 than 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 framing and/or grouping 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.

- `frame_style`, `frame_title` (default: nil) If defined, a frame will be drawn around the panel and subviews will be inset by 1. The attributes are identical to what is defined in the *FramedScreen class*. When using the predefined frame styles in the `gui` module, remember to `require` the gui module and prefix the identifier with `gui.`, e.g. `gui.GREY_LINE_FRAME`.

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:

label_text The optional text label displayed before the editable text.

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)`.

on_submit2 Shift-Enter key callback; if set the field will handle the key and call `on_submit2(text)`.

key If specified, the field is disabled until this key is pressed. Must be given as a string.

key_sep If specified, will be used to customize how the activation key is displayed. See `token.key_sep` in the `Label` documentation below.

modal Whether the `EditField` should prevent input from propagating to other widgets while it has focus. You can set this to `true`, for example, if you don't want a `List` widget to react to arrow keys while the user is editing.

ignore_keys If specified, must be a list of key names that the edit field should ignore. This is useful if you have plain string characters that you want to use as hotkeys (like +).

An `EditField` will only read and process text input if it has keyboard focus. It will automatically acquire keyboard focus when it is added as a subview to a parent that has not already granted keyboard focus to another widget. If you have more than one `EditField` on a screen, you can select which has focus by calling `setFocus(true)` on the field object.

If an activation `key` is specified, the `EditField` will manage its own focus. It will start in the unfocused state, and pressing the activation key will acquire keyboard focus. Pressing the Enter key will release keyboard focus and then call the `on_submit` callback. Pressing the Escape key will also release keyboard focus, but first it will restore the text that was displayed before the `EditField` gained focus and then call the `on_change` callback.

The `EditField` cursor can be moved to where you want to insert/remove text. You can click where you want the cursor to move or you can use any of the following keyboard hotkeys:

- Left/Right arrow: move the cursor one character to the left or right.
- Ctrl-Left/Right arrow: move the cursor one word to the left or right.
- Alt-Left/Right arrow: move the cursor to the beginning/end of the text.

Scrollbar class

This Widget subclass implements mouse-interactive scrollbars whose bar sizes represent the amount of content currently visible in an associated display widget (like a *Label class* or a *List class*). By default they are styled like scrollbars used in the vanilla DF help screens, but they are configurable.

Scrollbars have the following attributes:

fg Specifies the pen for the scroll icons and the active part of the bar. Default is `COLOR_LIGHTGREEN`.

bg Specifies the pen for the background part of the scrollbar. Default is `COLOR_CYAN`.

on_scroll A callback called when the scrollbar is scrolled. If the scrollbar is clicked, the callback will be called with one of the following string parameters: "up_large", "down_large", "up_small", or "down_small". If the scrollbar is dragged, the callback will be called with the value that `top_elem` should be set to on the next call to `update()` (see below).

The Scrollbar widget implements the following methods:

- `scrollbar:update(top_elem, elems_per_page, num_elems)`

  Updates the info about the widget that the scrollbar is paired with. The `top_elem` param is the (one-based) index of the first visible element. The `elems_per_page` param is the maximum number of elements that can be shown at one time. The `num_elems` param is the total number of elements that the paried widget can scroll through. If `elems_per_page` or `num_elems` is not specified, the most recently specified value for these parameters is used. The scrollbar will adjust its scrollbar size and position according to the values passed to this function.

Clicking on the arrows at the top or the bottom of a scrollbar will scroll an associated widget by a small amount. Clicking on the unfilled portion of the scrollbar above or below the filled area will scroll by a larger amount in that direction. The amount of scrolling done in each case in determined by the associated widget, and after scrolling is complete, the associated widget must call `scrollbar:update()` with updated new display info.

You can click and drag the scrollbar to scroll to a specific spot, or you can click and hold on the end arrows or in the unfilled portion of the scrollbar to scroll multiple times, just like in a normal browser scrollbar. The speed of scroll events when the mouse button is held down is controlled by two global variables:

SCROLL_INITIAL_DELAY_MS The delay before the second scroll event.

SCROLL_DELAY_MS The delay between further scroll events.

The defaults are 300 and 20, respectively, but they can be overridden by the user in their `dfhack-config/init/dfhack.init` file, for example:

```
:lua require('gui.widgets').SCROLL_DELAY_MS = 100
```

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. The table should map keys to the number of lines to scroll as positive or negative integers or one of the keywords supported by the `scroll` method. The default is up/down arrows scrolling by one line and page up/down scrolling by one page.

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 starts with '()', the token is formatted as `text..' ('..binding..sep:sub(2)`. 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.
- `label:scroll(nlines)`

  This method takes the number of lines to scroll as positive or negative integers or one of the following keywords: `+page`, `-page`, `+halfpage`, or `-halfpage`. It returns the number of lines that were actually scrolled (negative for scrolling up).

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 or mouse click.

It has the following attributes:

key The hotkey keycode to display, e.g. `'CUSTOM_A'`.

key_sep If specified, will be used to customize how the activation key is displayed. See `token.key_sep` in the `Label` documentation.

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 or the label is clicked.

CycleHotkeyLabel class

This Label subclass represents a group of related options that the user can cycle through by pressing a specified hotkey or clicking on the text.

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. You can click on a list item to call the `on_submit` callback for that item.

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 or mouse click callback; if specified, the list reacts to the key/click and calls the callback as `on_submit(index,choice)`.

on_submit2 Shift-Enter key or shift-mouse click callback; if specified, the list reacts to the key/click 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.

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:getIdxUnderMouse()`

  Returns the index of the list item under the mouse cursor, or nothing if the list is empty or the mouse is not over a list item.

- `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.

edit_ignore_keys If specified, will be passed to the filter edit field as its `ignore_keys` attribute.

edit_on_char If specified, will be passed to the filter edit field as its `on_char` attribute.

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.

6.6.5 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 impassable tiles impassable 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 animations. It looks over the 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 power production 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
```

(continues on next page)

(continued from previous page)

```
    {{x=0,y=0,15,7,0,0}} -- second frame, same
    }
  }
```

Or with auto_gears:

```
require('plugins.building-hacks').registerBuilding{name="BONE_GRINDER",
  consume=15,
  auto_gears=true
  }
```

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 buildings 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
```

(continues on next page)

(continued from previous page)

```
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. `onReactionCompleting(reaction,reaction_product,unit,input_items,input_reagents, output_items,call_native)`

   Is called once per reaction product, before the reaction has a chance to call native code for item creation. Setting `call_native.value=false` cancels further processing: no items are created and `onReactionComplete` is not called.

2. `onReactionComplete(reaction,reaction_product,unit,input_items,input_reagents, output_items)`

   Is called once per reaction product, when reaction finishes and has at least one product.

3. `onItemContaminateWound(item,unit,wound,number1,number2)`

   Is called when item tries to contaminate wound (e.g. stuck in).

4. `onProjItemCheckMovement(projectile)`

   Is called when projectile moves.

5. `onProjItemCheckImpact(projectile,somebool)`

   Is called when projectile hits something.

6. `onProjUnitCheckMovement(projectile)`

   Is called when projectile moves.

7. `onProjUnitCheckImpact(projectile,somebool)`

   Is called when projectile hits something.

8. `onWorkshopFillSidebarMenu(workshop,callnative)`

   Is called when viewing a workshop in 'q' mode, to populate reactions, useful for custom viewscreens for shops.

9. `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 canceled 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 onReactionCompleting; 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. Useful 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("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 a subset of the 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 the 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
    )
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
)
```

6.6.6 Scripts

Any files with the `.lua` extension placed into the `hack/scripts` folder are automatically made available 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: 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.

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 rendered (or embedded) *DFHack documentation system* for 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 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`.

6.7 DFHack overlay dev guide

This guide walks you through how to build overlay widgets and register them with the *overlay* framework for injection into Dwarf Fortress viewscreens.

6.7.1 Why would I want to create an overlay widget?

There are both C++ and Lua APIs for creating viewscreens and drawing to the screen. If you need very specific low-level control, those APIs might be the right choice for you. However, here are some reasons you might want to implement an overlay widget instead:

1. **You can draw directly to an existing viewscreen instead of creating an** entirely new screen on the viewscreen stack. This allows the original viewscreen to continue processing uninterrupted and keybindings bound to that viewscreen will continue to function. This was previously only achievable by C++ plugins.
2. **You'll get a free UI for enabling/disabling your widget and repositioning it** on the screen. Widget state is saved for you and is automatically restored when the game is restarted.
3. **You don't have to manage the C++ interposing logic yourself and can focus on** the business logic, writing purely in Lua if desired.

In general, if you are writing a plugin or script and have anything you'd like to add to an existing screen (including live updates of map tiles while the game is unpaused), an overlay widget is probably your easiest path to get it done. If your plugin or script doesn't otherwise need to be enabled to function, using the overlay allows you to avoid writing any of the enable management code that would normally be required for you to show info in the UI.

6.7.2 Overlay widget API

Overlay widgets are Lua classes that inherit from `overlay.OverlayWidget` (which itself inherits from *widgets.Widget*). The regular `onInput(keys)`, `onRenderFrame(dc, frame_rect)`, and `onRenderBody(dc)` functions work as normal, and they are called when the viewscreen that the widget is associated with does its usual input and render processing. The widget gets first dibs on input processing. If a widget returns `true` from its `onInput()` function, the viewscreen will not receive the input.

Overlay widgets can contain other Widgets and be as simple or complex as you need them to be, just like you're building a regular UI element.

There are a few extra capabilities that overlay widgets have that take them beyond your everyday `Widget`:

- **If an `overlay_onupdate(viewscreen)` function is defined, it will be called** just after the associated viewscreen's `logic()` function is called (i.e. a "tick" or a (non-graphical) "frame"). For hotspot widgets, this function will also get called after the top viewscreen's `logic()` function is called, regardless of whether the widget is associated with that viewscreen. If this function returns `true`, then the widget's `overlay_trigger()` function is immediately called. Note that the `viewscreen` parameter will be `nil` for hotspot widgets that are not also associated with the current viewscreen.
- **If an `overlay_trigger()` function is defined, will be called when the** widget's `overlay_onupdate` callback returns true or when the player uses the CLI (or a keybinding calling the CLI) to trigger the widget. The function must return either `nil` or the `gui.Screen` object that the widget code has allocated, shown, and now owns. Hotspot widgets will receive no callbacks from unassociated viewscreens until the returned screen is dismissed. Unbound hotspot widgets **must** allocate a Screen with this function if they want to react to the `onInput()` feed or be rendered. The widgets owned by the overlay framework must not be attached to that new screen, but the returned screen can instantiate and configure any new views that it wants to.

If the widget can take up a variable amount of space on the screen, and you want the widget to adjust its position according to the size of its contents, you can modify `self.frame.w` and `self.frame.h` at any time – in `init()` or in any of the callbacks – to indicate a new size. The overlay framework will detect the size change and adjust the widget position and layout.

If you don't need to dynamically resize, just set `self.frame.w` and `self.frame.h` once in `init()`.

Widget attributes

The `overlay.OverlayWidget` superclass defines the following class attributes:

- **`name`** This will be filled in with the display name of your widget, in case you have multiple widgets with the same implementation but different configurations.
- **`default_pos` (default: `{x=-2, y=-2}`)** Override this attribute with your desired default widget position. See the *overlay* docs for information on what positive and negative numbers mean for the position. Players can change the widget position at any time via the *overlay position* command, so don't assume that your widget will always be at the default position.
- **`viewscreens` (default: `{}`)** The list of viewscreens that this widget should be associated with. When one of these viewscreens is on top of the viewscreen stack, your widget's callback functions for update, input, and render will be interposed into the viewscreen's call path. The name of the viewscreen is the name of the DFHack class that represents the viewscreen, minus the `viewscreen_` prefix and `st` suffix. For example, the fort mode main map viewscreen would be `dwarfmode` and the adventure mode map viewscreen would be `dungeonmode`. If there is only one viewscreen that this widget is associated with, it can be specified as a string instead of a list of strings with a single element.
- **`hotspot` (default: `false`)** If set to `true`, your widget's `overlay_onupdate` function will be called whenever the *overlay* plugin's `plugin_onupdate()` function is called (which corresponds to one call per call to the current top viewscreen's `logic()` function). This call to `overlay_onupdate` is in addition to any calls initiated from associated interposed viewscreens and will come after calls from associated viewscreens.
- **`overlay_only` (default: `false`)** If set to `true`, no widget frame will be drawn in *gui/overlay* for drag and drop repositioning. Overlay widgets that don't have a "widget" to reposition should set this to `true`.
- **`overlay_onupdate_max_freq_seconds` (default: `5`)** This throttles how often a widget's `overlay_onupdate` function can be called (from any source). Set this to the largest amount of time (in seconds) that your widget can take to react to changes in information and not annoy the player. Set to 0 to be called at the maximum rate. Be aware that running more often than you really need to will impact game FPS, especially if your widget can run while the game is unpaused.

Registering a widget with the overlay framework

Anywhere in your code after the widget classes are declared, define a table named `OVERLAY_WIDGETS`. The keys are the display names for your widgets and the values are the widget classes. For example, the *dwarfmonitor* widgets are declared like this:

```
OVERLAY_WIDGETS = {
    cursor=CursorWidget,
    date=DateWidget,
    misery=MiseryWidget,
    weather=WeatherWidget,
}
```

When the *overlay* plugin is enabled, it scans all plugins and scripts for this table and registers the widgets on your behalf. The widget is enabled if it was enabled the last time the *overlay* plugin was loaded and the widget's position is restored according to the state saved in the `dfhack-config/overlay.json` file.

The overlay framework will instantiate widgets from the named classes and own the resulting objects. The instantiated widgets must not be added as subviews to any other View, including the Screen views that can be returned from the `overlay_trigger()` function.

6.7.3 Widget example 1: adding text to a DF screen

This is a simple widget that displays a message at its position. The message text is retrieved from the host script or plugin every ~20 seconds or when the `AltZ` hotkey is hit:

```
local overlay = require('plugins.overlay')
local widgets = require('gui.widgets')

MessageWidget = defclass(MessageWidget, overlay.OverlayWidget)
MessageWidget.ATTRS{
    default_pos={x=5,y=-2},
    viewscreens={'dwarfmode', 'dungeonmode'},
    overlay_onupdate_max_freq_seconds=20,
}

function MessageWidget:init()
    self.label = widgets.Label{text=''}
    self:addviews{self.label}
end

function MessageWidget:overlay_onupdate()
    local text = getImportantMessage() -- defined in the host script/plugin
    self.label:setText(text)
    self.frame.w = #text
end

function MessageWidget:onInput(keys)
    if keys.CUSTOM_ALT_Z then
        self:overlay_onupdate()
        return true
    end
end

OVERLAY_WIDGETS = {message=MessageWidget}
```

6.7.4 Widget example 2: highlighting artifacts on the live game map

This widget is not rendered at its "position" at all, but instead monitors the map and overlays information about where artifacts are located. Scanning for which artifacts are visible on the map can slow, so that is only done every 10 seconds to avoid slowing down the entire game on every frame.

```
local overlay = require('plugins.overlay')
local widgets = require('gui.widgets')

ArtifactRadarWidget = defclass(ArtifactRadarWidget, overlay.OverlayWidget)
ArtifactRadarWidget.ATTRS{
    viewscreens={'dwarfmode', 'dungeonmode'},
    overlay_onupdate_max_freq_seconds=10,
}

function ArtifactRadarWidget:overlay_onupdate()
    self.visible_artifacts_coords = getVisibleArtifactCoords()
```

(continues on next page)

(continued from previous page)

```
end

function ArtifactRadarWidget:onRenderFrame()
    for _,pos in ipairs(self.visible_artifacts_coords) do
        -- highlight tile at given coordinates
    end
end

OVERLAY_WIDGETS = {radar=ArtifactRadarWidget}
```

6.7.5 Widget example 3: corner hotspot

This hotspot reacts to mouseover events and launches a screen that can react to input events. The hotspot area is a 2x2 block near the lower right corner of the screen (by default, but the player can move it wherever).

```
local overlay = require('plugins.overlay')
local widgets = require('gui.widgets')

HotspotMenuWidget = defclass(HotspotMenuWidget, overlay.OverlayWidget)
HotspotMenuWidget.ATTRS{
    default_pos={x=-3,y=-3},
    frame={w=2, h=2},
    hotspot=true,
    viewscreens='dwarfmode',
    overlay_onupdate_max_freq_seconds=0, -- check for mouseover every tick
}

function HotspotMenuWidget:init()
    -- note this label only gets rendered on the associated viewscreen
    -- (dwarfmode), but the hotspot is active on all screens
    self:addviews{widgets.Label{text={'!!', NEWLINE, '!!'}}}
    self.mouseover = false
end

function HotspotMenuWidget:overlay_onupdate()
    local hasMouse = self:getMousePos()
    if hasMouse and not self.mouseover then -- only trigger on mouse entry
        self.mouseover = true
        return true
    end
    self.mouseover = hasMouse
end

function HotspotMenuWidget:overlay_trigger()
    return MenuScreen{hotspot_frame=self.frame}:show()
end

OVERLAY_WIDGETS = {menu=HotspotMenuWidget}

MenuScreen = defclass(MenuScreen, gui.Screen)
MenuScreen.ATTRS{
```

(continues on next page)

(continued from previous page)

```
    focus_path='hotspot/menu',
    hotspot_frame=DEFAULT_NIL,
}

function MenuScreen:init()
    self.mouseover = false

    -- derrive the menu frame from the hotspot frame so it
    -- can appear in a nearby location
    local frame = copyall(self.hotspot_frame)
    -- ...

    self:addviews{
        widgets.ResizingPanel{
            autoarrange_subviews=true,
            frame=frame,
            frame_style=gui.GREY_LINE_FRAME,
            frame_background=gui.CLEAR_PEN,
            subviews={
                -- ...
            },
        },
    },
    }
end

function MenuScreen:onInput(keys)
    if keys.LEAVESCREEN then
        self:dismiss()
        return true
    end
    return self:inputToSubviews(keys)
end

function MenuScreen:onRenderFrame(dc, rect)
    self:renderParent()
end
```

6.8 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 into 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:

6.8.1 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
  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.

6.8.2 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
            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 …

6.9 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

6.9.1 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 tools 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.

6.9.2 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.

6.9.3 Windows-specific tools

Some people have used Cheat Engine for research in the past.

6.10 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

6.10.1 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.

6.10.2 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.

6.10.3 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.

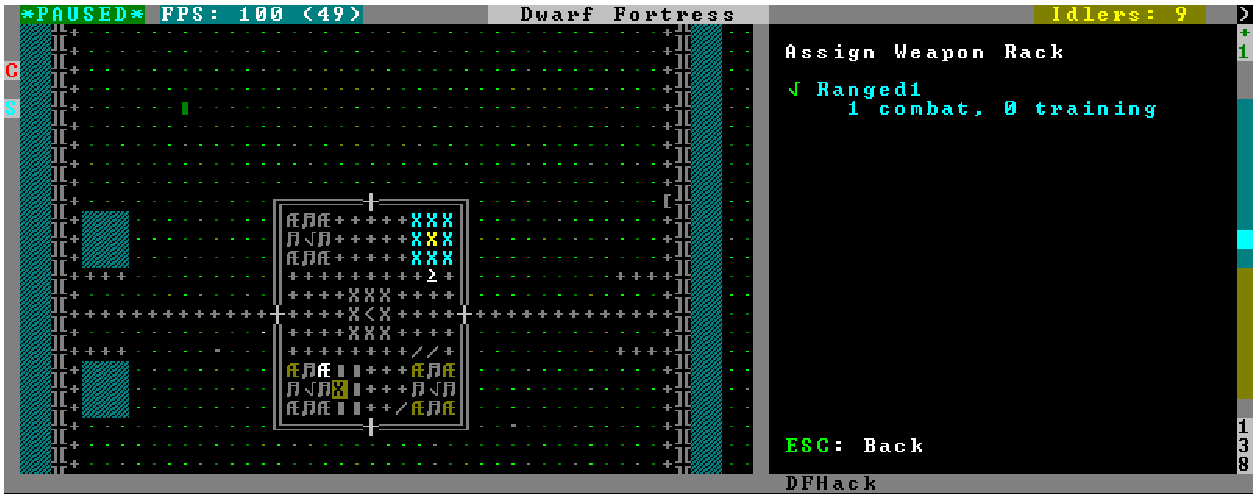
Contrary to the common misconception, all these uses are controlled by the *Individual Equipment* usage flag. The *Squad Equipment* flag is actually intended for ammo, but the game does even less in that area than for armor and weapons. This plugin implements the following rules almost from scratch:

- 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.



```
Assign Weapon Rack

√ Ranged1
    1 combat, 0 training

ESC: Back
```

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.

6.11 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

6.11.1 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.

6.11.2 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

6.11.3 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)`

6.12 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

6.12.1 DFHack 0.47.05-r8

New Plugins

- *overlay*: plugin is transformed from a single line of text that runs *gui/launcher* on click to a fully-featured overlay injection framework. It now houses a popup menu for keybindings relevant to the current DF screen, all the widgets previously provided by *dwarfmonitor* (e.g. the current date and number of happy/unhappy dwarves), the overlay that highlights suspended buildings when you pause, and others. See *DFHack overlay dev guide* for details.

New Scripts

- *gui/overlay*: configuration interface for the DFHack overlays and overlay widgets. includes a click-and-drag interface for repositioning widgets!

Fixes

- Core: ensure `foo.init` always runs before `foo.*.init` (e.g. `dfhack.init` should always run before `dfhack.something.init`)
- *autofarm*: flush output so status text is visible immediately after running the command
- *autolabor*, *autohauler*: properly handle jobs 241, 242, and 243
- ***automaterial*:**
  - fix the cursor jumping up a z level when clicking quickly after box select
  - fix rendering errors with box boundary markers
- *buildingplan*: fix crash when canceling out of placement mode for a building with planning mode enabled and subsequently attempting to place a building that does not have planning mode enabled and that has no pertinent materials available
- *dwarf-op*: fixed error when matching dwarves by name
- *gui/create-item*: prevent materials list filter from intercepting sublist hotkeys
- *gui/gm-unit*: fixed behavior of `+` and `-` to adjust skill values instead of populating the search field
- *hotkeys*: correctly detect hotkeys bound to number keys, F11, and F12
- *labormanager*: associate quern construction with the correct labor
- *mousequery*: fix the cursor jumping up z levels sometimes when using TWBT
- *tiletypes*: no longer resets dig priority to the default when updating other properties of a tile
- ***warn-stealers*:**
  - register callback with correct event name so that units entering the map are detected
  - announce thieving creatures that spawn already revealed
  - cache unit IDs instead of unit objects to avoid referencing stale pointers
- *workorder*: fix interpretation of json-specified orders that set the `item_type` field
- **`EventManager`:**
  - fix a segmentation fault with the `REPORT` event

  - fix the `JOB_STARTED` event only sending events to the first handler listed instead of all registered handlers

Misc Improvements

- **UX:**
  - List widgets now have mouse-interactive scrollbars
  - You can now hold down the mouse button on a scrollbar to make it scroll multiple times.
  - You can now drag the scrollbar up and down to scroll to a specific spot
- *autolabor*, *autohauler*: refactored to use DFHack's messaging system for info/debug/trace messages
- ***blueprint***:
  - new `--smooth` option for recording all smoothed floors and walls instead of just the ones that require smoothing for later carving
  - record built constructions in blueprints
  - record stockpile/building/zone names in blueprints
  - record room sizes in blueprints
  - generate meta blueprints to reduce the number of blueprints you have to apply
  - support splitting the output file into phases grouped by when they can be applied
  - when splitting output files, number them so they sort into the order you should apply them in
- *dig*: new `-z` option for digtype to restrict designations to the current z-level and down
- *dwarfmonitor*: widgets have been ported to the overlay framework and can be enabled and configured via the *gui/overlay* UI
- *gui/blueprint*: support new blueprint phases and options
- *gui/cp437-table*: new global keybinding for the clickable on-screen keyboard for players with keyboard layouts that prevent them from using certain keys: Ctrl-Shift-K
- *gui/create-item*: restrict materials to those normally allowed by the game by default, introduce new `--unrestricted` option for full freedom in choosing materials
- *gui/launcher*: show help for commands that start with ':' (like `:lua`)
- *gui/quantum*: add option to allow corpses and refuse in your quantum stockpile
- ***hotkeys***:
  - hotkey screen has been transformed into an interactive *overlay* widget that you can bring up by moving the mouse cursor over the hotspot (in the upper left corner of the screen by default). Enable/disable/reposition the hotspot in the *gui/overlay* UI. Even if the hotspot is disabled, the menu can be brought up at any time with the Ctrl-Shift-C hotkey.
  - now supports printing active hotkeys to the console with `hotkeys list`
- ***ls***:
  - indent tag listings and wrap them in the rightmost column for better readability
  - new `--exclude` option for hiding matched scripts from the output. this can be especially useful for modders who don't want their mod scripts to be included in `ls` output.

- *modtools/create-unit*: better unit naming, more argument checks, assign nemesis save data for units without civilization so they can be properly saved when offloaded
- *orders*: replace shell craft orders in the standard orders list you get with `orders import library/basic` with orders for shell leggings. They have a slightly higher trade price. Also, "shleggings" is just hilarious.
- *Quickfort blueprint library*: improved layout of marksdwarf barracks in the example Dreamfort blueprints
- ***spectate*:**
  - new `auto-unpause` option for auto-dismissal of announcement pause events (e.g. sieges).
  - new `auto-disengage` option for auto-disengagement of plugin through player interaction whilst unpaused.
  - new `tick-threshold` option for specifying the maximum time to follow the same dwarf
  - new `animals` option for sometimes following animals
  - new `hostiles` option for sometimes following hostiles
  - new `visiting` option for sometimes following visiting merchants, diplomats or plain visitors
  - added persistent configuration of the plugin settings
- *unsuspend*: new *overlay* for displaying status of suspended buildings (functionality migrated from removed *resume* plugin)

Removed

- *gui/create-item*: removed `--restricted` option. it is now the default behavior
- *resume*: functionality (including suspended building overlay) has moved to *unsuspend*

API

- Constructions module: added `insert()` to insert constructions into the game's sorted list.
- MiscUtils: added the following string transformation functions (refactored from `uicommon.h`): `int_to_string`, `ltrim`, `rtrim`, and `trim`; added `string_to_int`
- **Units module:**
  - added new predicates for:
  - `isUnitInBox()`
  - `isAnimal()`
  - `isVisiting()` any visiting unit (diplomat, merchant, visitor)
  - `isVisitor()` ie. not merchants or diplomats
  - `isInvader()`
  - `isDemon()` returns true for unique/regular demons
  - `isTitan()`
  - `isMegabeast()`
  - `isGreatDanger()` returns true if unit is a demon, titan, or megabeast
  - `isSemiMegabeast()`
  - `isNightCreature()`

  - `isDanger()` returns true if is a 'GreatDanger', semi-megabeast, night creature, undead, or invader
  - modified predicates:
  - `isUndead()` now optionally ignores vampires instead of always ignoring vampires
  - `isCitizen()` now optionally ignores insane citizens instead of always ignoring insane citizens
- `Gui::anywhere_hotkey`: for plugin commands bound to keybindings that can be invoked on any screen
- `Gui::autoDFAnnouncement`, `Gui::pauseRecenter`: added functionality reverse-engineered from announcement code
- `Gui::revealInDwarfmodeMap`: Now enforce valid view bounds when pos invalid, add variant accepting x, y, z
- `Lua::Push()`: now handles maps with otherwise supported keys and values
- `Lua::PushInterfaceKeys()`: transforms viewscreen `feed()` keys into something that can be interpreted by lua-based widgets

Internals

- Constructions module: `findAtTile` now uses a binary search intead of a linear search
- MSVC warning level upped to /W3, and /WX added to make warnings cause compilations to fail.

Lua

- Lua mouse events now conform to documented behavior in *DFHack Lua API Reference* – `_MOUSE_L_DOWN` will be sent exactly once per mouse click and `_MOUSE_L` will be sent repeatedly as long as the button is held down. Similarly for right mouse button events.
- `dfhack.constructions.findAtTile()`: exposed preexisting function to Lua.
- `dfhack.constructions.insert()`: exposed new function to Lua.
- `gui.Screen.show()`: now returns `self` as a convenience
- `gui.View.getMousePos()` now takes an optional `ViewRect` parameter in case the caller wants to get the mouse pos relative to a rect that is not the frame_body (such as the frame_rect that includes the frame itself)
- `widgets.EditField`: now allows other widgets to process characters that the `on_char` callback rejects.
- `widgets.FilteredList`: now provides a useful default search key for list items made up of text tokens instead of plain text
- `widgets.HotkeyLabel`: now ignores mouse clicks when `on_activate` is not defined
- **`widgets.List`:**
  - new `getIdxUnderMouse()` function for detecting the list index under the active mouse cursor. this allows for "selection follows mouse" behavior
  - shift-clicking now triggers the `submit2` attribute function if it is defined
- `widgets.Panel`: new `frame_style` and `frame_title` attributes for drawing frames around groups of widgets
- `widgets.ResizingPanel`: now accounts for frame inset when calculating frame size
- `widgets.Scrollbar`: new scrollbar widget that can be paired with an associated scrollable widget. Integrated with `widgets.Label` and `widgets.List`.

Structures

- `general_refst`: type virtual union member for `ITEM_GENERAL`
- `historical_figure_info.T_reputation.unk_2c`: identify `year` + `year_ticks`
- `itemst`: identify two vmethods related to adding thread improvements to items made of cloth, and label several previously unknown return types
- `proj_magicst`: correct structure fields (to match 40d)
- `unit_action_type_group`: added enum and tagged `unit_action_type` entries with its groups for DFHack's new action timer API.
- `world`: identify type of a vector (still not known what it's for, but it's definitely an item vector)

Documentation

- *DFHack overlay dev guide*: documentation and guide for injecting functionality into DF viewscreens from Lua scripts and creating interactive overlay widgets
- `dfhack.gui.revealInDwarfmodeMap`: document `center` bool for Lua API

6.12.2 DFHack 0.47.05-r7

New Plugins

- *autobutcher*: split off from *zone* into its own plugin. Note that to enable, the command has changed from `autobutcher start` to `enable autobutcher`.
- *autonestbox*: split off from *zone* into its own plugin. Note that to enable, the command has changed from `autonestbox start` to `enable autonestbox`.
- *overlay*: display a "DFHack" button in the lower left corner that you can click to start the new GUI command launcher. The *dwarfmonitor* weather display had to be moved to make room for the button. If you are seeing the weather indicator rendered over the overlay button, please remove the `dfhack-config/dwarfmonitor.json` file to fix the weather indicator display offset.

New Scripts

- *gui/kitchen-info*: adds more info to the Kitchen screen
- *gui/launcher*: in-game command launcher with autocomplete, history, and context-sensitive help
- *gui/workorder-details*: adjusts work orders' input item, material, traits
- *max-wave*: dynamically limit the next immigration wave, can be set to repeat
- *pop-control*: persistent per fortress population cap, *hermit*, and *max-wave* management
- *warn-stealers*: warn when creatures that may steal your food, drinks, or items become visible

New Internal Commands

- *tags*: new built-in command to list the tool category tags and their definitions. tags associated with each tool are visible in the tool help and in the output of *ls*.

Fixes

- *autochop*: designate largest trees for chopping first, instead of the smallest
- *devel/query*: fixed error when –tile is specified
- *dig-now*: Fix direction of smoothed walls when adjacent to a door or floodgate
- *dwarf-op*: fixed error when applying the Miner job to dwarves
- *emigration*: fix emigrant logic so unhappy dwarves leave as designed
- *gui/gm-unit*: allow + and - to adjust skill values as intended instead of letting the filter intercept the characters
- *gui/unit-info-viewer*: fix logic for displaying undead creature names
- *gui/workflow*: restore functionality to the add/remove/order hotkeys on the workflow status screen
- *modtools/moddable-gods*: fixed an error when assigning spheres
- *quickfort*: *Dreamfort* blueprint set: declare the hospital zone before building the coffer; otherwise DF fails to stock the hospital with materials
- *view-item-info*: fixed a couple errors when viewing items without materials
- `dfhack.buildings.findCivzonesAt`: no longer return duplicate civzones after loading a save with existing civzones
- `dfhack.run_script`: ensure the arguments passed to scripts are always strings. This allows other scripts to call `run_script` with numeric args and it won't break parameter parsing.
- `job.removeJob()`: ensure jobs are removed from the world list when they are canceled

Misc Improvements

- History files: `dfhack.history`, `tiletypes.history`, `lua.history`, and `liquids.history` have moved to the `dfhack-config` directory. If you'd like to keep the contents of your current history files, please move them to `dfhack-config`.
- Init scripts: `dfhack.init` and other init scripts have moved to `dfhack-config/init/`. If you have customized your `dfhack.init` file and want to keep your changes, please move the part that you have customized to the new location at `dfhack-config/init/dfhack.init`. If you do not have changes that you want to keep, do not copy anything, and the new defaults will be used automatically.
- **UX:**
  - You can now move the cursor around in DFHack text fields in `gui/` scripts (e.g. *gui/blueprint*, *gui/quickfort*, or *gui/gm-editor*). You can move the cursor by clicking where you want it to go with the mouse or using the Left/Right arrow keys. Ctrl+Left/Right will move one word at a time, and Alt+Left/Right will move to the beginning/end of the text.
  - You can now click on the hotkey hint text in many `gui/` script windows to activate the hotkey, like a button. Not all scripts have been updated to use the clickable widget yet, but you can try it in *gui/blueprint* or *gui/quickfort*.

  - Label widget scroll icons are replaced with scrollbars that represent the percentage of text on the screen and move with the position of the visible text, just like web browser scrollbars.

- ***devel/query*:**
  - inform the user when a query has been truncated due to `--maxlength` being hit.
  - increased default maxlength value from 257 to 2048
- *do-job-now*: new global keybinding for boosting the priority of the jobs associated with the selected building/work order/unit/item etc.: Alt-N
- *dwarf-op*: replaces [a b c] option lists with a,b,c option lists
- *gui/gm-unit*: don't clear the list filter when you adjust a skill value
- ***gui/quickfort*:**
  - better formatting for the generated manager orders report
  - you can now click on the map to move the blueprint anchor point to that tile instead of having to use the cursor movement keys
  - display an error message when the blueprints directory cannot be found
- *gui/workorder-details*: new keybinding on the workorder details screen: `D`
- *keybinding*: support backquote (`) as a hotkey (and assign the hotkey to the new *gui/launcher* interface)
- *ls*: can now filter tools by substring or tag. note that dev scripts are hidden by default. pass the `--dev` option to show them.
- ***manipulator*:**
  - add a library of useful default professions
  - move professions configuration from `professions/` to `dfhack-config/professions/` to keep it together with other dfhack configuration. If you have saved professions that you would like to keep, please manually move them to the new folder.
- *orders*: added useful library of manager orders. see them with `orders list` and import them with, for example, `orders import library/basic`
- *prioritize*: new `defaults` keyword to prioritize the list of jobs that the community agrees should generally be prioritized. Run `prioritize -a defaults` to try it out in your fort!
- *prospector*: add new `--show` option to give the player control over which report sections are shown. e.g. `prospect all --show ores` will just show information on ores.
- ***quickfort*:**
  - *Dreamfort* blueprint set improvements: set traffic designations to encourage dwarves to eat cooked food instead of raw ingredients
  - library blueprints are now included by default in `quickfort list` output. Use the new `--useronly` (or just `-u`) option to filter out library bluerpints.
  - better error message when the blueprints directory cannot be found
- *seedwatch*: `seedwatch all` now adds all plants with seeds to the watchlist, not just the "basic" crops.
- `materials.ItemTraitsDialog`: added a default `on_select`-handler which toggles the traits.

Removed

- *fix/build-location*: The corresponding DF bug (5991) was fixed in DF 0.40.05
- *fix/diplomats*: DF bug 3295 fixed in 0.40.05
- *fix/fat-dwarves*: DF bug 5971 fixed in 0.40.05
- *fix/feeding-timers*: DF bug 2606 is fixed in 0.40.12
- *fix/merchants*: DF bug that prevents humans from making trade agreements has been fixed
- *gui/assign-rack*: No longer useful in current DF versions
- *gui/hack-wish*: Replaced by *gui/create-item*
- *gui/no-dfhack-init*: No longer useful since players don't have to create their own `dfhack.init` files anymore

API

- Removed "egg" ("eggy") hook support (Linux only). The only remaining method of hooking into DF is by interposing SDL calls, which has been the method used by all binary releases of DFHack.
- Removed `Engravings` module (C++-only). Access `world.engravings` directly instead.
- Removed `Notes` module (C++-only). Access `ui.waypoints.points` directly instead.
- Removed `Windows` module (C++-only) - unused.
- `Constructions` module (C++-only): removed `t_construction`, `isValid()`, `getCount()`, `getConstruction()`, and `copyConstruction()`. Access `world.constructions` directly instead.
- `Gui::getSelectedItem()`, `Gui::getAnyItem()`: added support for the artifacts screen
- `Units::teleport()`: now sets `unit.idle_area` to discourage units from walking back to their original location (or teleporting back, if using *fastdwarf*)

Lua

- Added `dfhack.screen.hideGuard()`: exposes the C++ `Screen::Hide` to Lua
- History: added `dfhack.getCommandHistory(history_id, history_filename)` and `dfhack.addCommandToHistory(history_id, history_filename, command)` so gui scripts can access a commandline history without requiring a terminal.
- `helpdb`: database and query interface for DFHack tool help text
- `tile-material`: fix the order of declarations. The `GetTileMat` function now returns the material as intended (always returned nil before). Also changed the license info, with permission of the original author.
- `utils.df_expr_to_ref()`: fixed some errors that could occur when navigating tables
- `widgets.CycleHotkeyLabel`: clicking on the widget will now cycle the options and trigger `on_change()`. This also applies to the `ToggleHotkeyLabel` subclass.
- **`widgets.EditField`:**
  - new `onsubmit2` callback attribute is called when the user hits Shift-Enter.
  - new function: `setCursor(position)` sets the input cursor.
  - new attribute: `ignore_keys` lets you ignore specified characters if you want to use them as hotkeys

- `widgets.FilteredList`: new attribute: `edit_ignore_keys` gets passed to the filter EditField as `ignore_keys`
- `widgets.HotkeyLabel`: clicking on the widget will now call `on_activate()`.
- `widgets.Label`: `scroll` function now interprets the keywords `+page`, `-page`, `+halfpage`, and `-halfpage` in addition to simple positive and negative numbers.

Structures

- Eliminate all "anon_X" names from structure fields
- `army`: change `squads` vector type to `world_site_inhabitant`, identify `min_smell_trigger``+``max_odor_level``+``max_low_light_vision``+``sense_creature_classes`
- `cave_column_rectangle`: identify coordinates
- `cave_column`: identify Z coordinates
- `embark_profile`: identify reclaim fields, add missing pet_count vector
- `entity_population`: identify `layer_id`
- `feature`: identify "shiftCoords" vmethod, `irritation_level` and `irritation_attacks` fields
- `flow_guide`: identify "shiftCoords" vmethod
- `general_refst`: name parameters on `getLocation` and `setLocation` vmethods
- `general_ref_locationst`: name member fields
- `historical_entity`: confirm `hostility_level` and `siege_tier`
- `item`: identify method `notifyCreatedMasterwork` that is called when a masterwork is created.
- `language_name_type`: identify `ElfTree` and `SymbolArtifice` thru `SymbolFood`
- `misc_trait_type`: update auto-decrement markers, remove obsolete references
- `timed_event`: identify `layer_id`
- `ui_advmode`: identify several fields as containing coordinates
- `ui_build_selector`: identify `cur_walk_tag` and `min_weight_races``+``max_weight_races`
- `ui`: identify actual contents of `unk5b88` field, identify infiltrator references
- `unitst`: identify `histeventcol_id` field inside status2
- `viewscreen_barterst`: name member fields
- `viewscreen_tradegoodsst`: rename trade_reply `OffendedAnimal``+``OffendedAnimalAlt` to `OffendedBoth``+``OffendedAnimal`
- `world_site_inhabitant`: rename `outcast_id` and `founder_outcast_entity_id`, identify `interaction_id` and `interaction_effect_idx`

Documentation

- Added *DFHack modding guide*
- Group DFHack tools by *tag* so similar tools are grouped and easy to find
- Update all DFHack tool documentation (300+ pages) with standard syntax formatting, usage examples, and overall clarified text.

6.12.3 DFHack 0.47.05-r6

New Scripts

- *assign-minecarts*: automatically assign minecarts to hauling routes that don't have one
- *deteriorate*: combines, replaces, and extends previous *deteriorateclothes*, *deterioratecorpses*, and *deterioratefood* scripts.
- *gui/petitions*: shows petitions. now you can see which guildhall/temple you agreed to build!
- *gui/quantum*: point-and-click tool for creating quantum stockpiles
- *gui/quickfort*: shows blueprint previews on the live map so you can apply them interactively
- *modtools/fire-rate*: allows modders to adjust the rate of fire for ranged attacks

Fixes

- *build-now*: walls built above other walls can now be deconstructed like regularly-built walls
- ***eventful*:**
  - fix `eventful.registerReaction` to correctly pass `call_native` argument thus allowing canceling vanilla item creation. Updated related documentation.
  - renamed NEW_UNIT_ACTIVE event to UNIT_NEW_ACTIVE to match the `EventManager` event name
  - fixed UNIT_NEW_ACTIVE event firing too often
- *gui/dfstatus*: no longer count items owned by traders
- *gui/unit-info-viewer*: fix calculation/labeling of unit size
- `job.removeJob()`: fixes regression in DFHack 0.47.05-r5 where items/buildings associated with the job were not getting disassociated when the job is removed. Now *build-now* can build buildings and *gui/mass-remove* can cancel building deconstruction again
- `widgets.CycleHotkeyLabel`: allow initial option values to be specified as an index instead of an option value

Misc Improvements

- *build-now*: buildings that were just designated with *buildingplan* are now built immediately (as long as there are items available to build the buildings with) instead of being skipped until buildingplan gets around to doing its regular scan
- *caravan*: new `unload` command, fixes endless unloading at the depot by reconnecting merchant pack animals that were disconnected from their owners
- ***confirm***:
  - added a confirmation dialog for removing manager orders
  - allow players to pause the confirmation dialog until they exit the current screen
- *deteriorate*: new `now` command immediately deteriorates items of the specified types
- ***DFHack config file examples***:
  - refine food preparation orders so meal types are chosen intelligently according to the amount of meals that exist and the number of aviailable items to cook with
  - reduce required stock of dye for "Dye cloth" orders
  - fix material conditions for making jugs and pots
  - make wooden jugs by default to differentiate them from other stone tools. this allows players to more easily select jugs out with a properly-configured stockpile (i.e. the new `woodentools` alias)
- *list-agreements*: now displays translated guild names, worshipped deities, petition age, and race-appropriate professions (e.g. "Craftsdwarf" instead of "Craftsman")
- ***Quickfort keystroke alias reference***:
  - new aliases: `forbidsearch`, `permitsearch`, and `togglesearch` use the *search* plugin to alter the settings for a filtered list of item types when configuring stockpiles
  - new aliases: `stonetools` and `woodentools`. the `jugs` alias is deprecated. please use `stonetools` instead, which is the same as the old `jugs` alias.
  - new aliases: `usablehair`, `permitusablehair`, and `forbidusablehair` alter settings for the types of hair/wool that can be made into cloth: sheep, llama, alpaca, and troll. The `craftrefuse` aliases have been altered to use this alias as well.
  - new aliases: `forbidthread`, `permitthread`, `forbidadamantinethread`, `permitadamantinethread`, `forbidcloth`, `permitcloth`, `forbidadamantinecloth`, and `permitadamantinecloth` give you more control how adamantine-derived items are stored
- ***quickfort***:
  - *Dreamfort* blueprint set improvements: automatically create tavern, library, and temple locations (restricted to residents only by default), automatically associate the rented rooms with the tavern
  - *Dreamfort* blueprint set improvements: new design for the services level, including were-bitten hospital recovery rooms and an appropriately-themed interrogation room next to the jail! Also fits better in a 1x1 embark for minimalist players.
- *workorder*: a manager is no longer required for orders to be created (matching bevavior in the game itself)

Removed

- *deterioratecIothes*: please use `deteriorate --types=clothes` instead
- *deterioratecorpses*: please use `deteriorate --types=corpses` instead
- *deterioratefood*: please use `deteriorate --types=food` instead
- *devel/unforbidall*: please use *unforbid* instead. You can silence the output with `unforbid all --quiet`

API

- `word_wrap`: argument `bool collapse_whitespace` converted to enum `word_wrap_whitespace_mode mode`, with valid modes `WSMODE_KEEP_ALL`, `WSMODE_COLLAPSE_ALL`, and `WSMODE_TRIM_LEADING`.

Lua

- `gui.View`: all `View` subclasses (including all `Widgets`) can now acquire keyboard focus with the new `View:setFocus()` function. See docs for details.
- `materials.ItemTraitsDialog`: new dialog to edit item traits (where "item" is part of a job or work order or similar). The list of traits is the same as in vanilla work order conditions "`t` change traits".
- **`widgets.EditField`:**
  - the `key_sep` string is now configurable
  - can now display an optional string label in addition to the activation key
  - views that have an `EditField` subview no longer need to manually manage the `EditField` activation state and input routing. This is now handled automatically by the new `gui.View` keyboard focus subsystem.
- `widgets.HotkeyLabel`: the `key_sep` string is now configurable

Structures

- `art_image_elementst`: identify vmethod `markDiscovered` and second parameter for `getName2`
- `art_image_propertyst`: identify parameters for `getName`
- `building_handler`: fix vmethod `get_machine_hookup_list` parameters
- `vermin`: identify `category` field as new enum
- `world.unk_26a9a8`: rename to `allow_announcements`

6.12.4 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 config file examples***:
  - 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()`
- `Job::removeJob()`: use the job cancel vmethod graciously provided by The Toady One in place of a synthetic method derived from reverse engineering

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, `Panel`s 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 extensions*: document DFHack string extensions (`startswith()`, `endswith()`, `split()`, `trim()`, `wrap()`, and `escape_pattern()`)
- *Quickfort blueprint creation 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)`

6.12.5 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*, *tiletypes*: 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
- *Quickfort blueprint library*: 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 config file examples*: 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

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`

6.12.6 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. *Data 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*
- *Data 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*, *tiletypes*: 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 config file examples*: documentation for all of *Dreamfort*'s supporting files (useful for all forts, not just Dreamfort!)
- *Quickfort blueprint library*: updated dreamfort documentation and added screenshots

6.12.7 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*

6.12.8 DFHack 0.47.05-r1

Fixes

- *confirm*: stopped exposing alternate names when convicting units
- *prospector*: 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*

6.12.9 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

6.12.10 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

6.12.11 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 reference*: alias syntax and alias standard library documentation for *quickfort* blueprints
- *Quickfort blueprint library*: overview of the quickfort blueprint library

6.12.12 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 creation 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

6.12.13 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

6.12.14 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

6.12.15 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
- *unretire-anyone*: added `-dead` argument to revive and enable selection of a dead historical figure to use as an adventurer in adv mode

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

6.12.16 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)

6.12.17 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*)

6.12.18 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
- *prospector*: (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

6.12.19 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_civlistst`)
- `spoils_report`: new type, most fields identified
- **`viewscreen_civlistst`:**
  - split `unk_20` into 3 pointers
  - identified new pages
  - identified new messenger-related fields
- **`viewscreen_image_creatorst`:**
  - fixed layout
  - identified many fields
- `viewscreen_reportlistst`: 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

6.12.20 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

6.12.21 DFHack 0.44.11-beta2.1

Internals

- *stonesense*: fixed build

6.12.22 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. *cleaners*
- *gui/workflow*: fixed advanced constraint menu for crafts

API

- Added `Screen::Hide` to temporarily hide screens, like *command-prompt*

6.12.23 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_civlistst: fixed layout and identified many fields

6.12.24 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_civlistst: added a few new fields (incomplete)
- viewscreen_locationsst: identified edit_input

6.12.25 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*
- ***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
- `item-descriptions`: fixed several grammatical errors

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)

6.12.26 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
- *prospector*: 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)

6.12.27 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

6.12.28 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)

6.12.29 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

6.12.30 DFHack 0.44.09-alpha1

Fixes

- *dig*: stopped designating non-vein tiles (open space, trees, etc.)
- *labormanager*: fixed crash due to dig jobs targeting some unrevealed map blocks

6.12.31 DFHack 0.44.08-alpha1

Fixes

- *fix/dead-units*: fixed a bug that could remove some arriving (not dead) units

6.12.32 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

6.12.33 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 renaming:**
  - `world.reactions` to `world.reactions.reactions`
  - `world.reaction_categories` to `world.reactions.reaction_categories`

6.12.34 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

6.12.35 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`

6.12.36 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

6.12.37 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

6.12.38 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_reportlistst: fixed layout, added mission_id vector
- world.status: named missions vector

6.12.39 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

6.12.40 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

6.12.41 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`

CHAPTER

SEVEN

ABOUT DFHACK

These pages contain information about the general DFHack project.

7.1 List of authors

The following is a list of people who have contributed to DFHack, in alphabetical order.

If you should be here and aren't, please get in touch on IRC or the forums, or make a pull request!

Name	Github	Other
8Z	8Z	
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	
Cameron Ewell	Ozzatron	
Carter Bray	Qartar	
Chris Dombroski	cdombroski	
Chris Parsons	chrismdp	
Clayton Hughes		
Clément Vuchener	cvuchener	

continues on next page

Table 1 – continued from previous page

Name	Github	Other
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	
Kromtec	Kromtec	

continues on next page

Table 1 – continued from previous page

Name	Github	Other
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
Rafał Karczmarczyk	CarabusX	
Raidau	Raidau	

continues 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	

continues 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	

7.2 Licenses

DFHack is distributed under the Zlib license, with some MIT- and BSD-licensed components. These licenses protect your right to use DFHack for any purpose, distribute copies, and so on.

The core, plugins, scripts, and other DFHack code all use the ZLib license unless noted otherwise. By contributing to DFHack, authors release the contributed work under this license.

DFHack also draws on several external packages. Their licenses are summarised here and reproduced below.

Component	License	Copyright
DFHack	Zlib	(c) 2009-2012, Petr Mrázek
clsocket	BSD 3-clause	(c) 2007-2009, CarrierLabs, LLC.
dirent	MIT	(c) 2006, Toni Ronkko
JSON.lua	CC-BY-SA	(c) 2010-2014, Jeffrey Friedl
jsoncpp	MIT	(c) 2007-2010, Baptiste Lepilleur
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

7.2.1 Zlib License

See https://en.wikipedia.org/wiki/Zlib_License

```
This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any
damages arising from the use of this software.

Permission is granted to anyone to use this software for any
purpose, including commercial applications, and to alter it and
redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must
   not claim that you wrote the original software. If you use this
   software in a product, an acknowledgment in the product
   documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and
   must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source
   distribution.
```

7.2.2 MIT License

See https://en.wikipedia.org/wiki/MIT_License

```
Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

7.2.3 BSD Licenses

See https://en.wikipedia.org/wiki/BSD_licenses

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in
    the documentation and/or other materials provided with the
    distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

`linenoise` adds no further clauses.

`protobuf` adds the following clause:

```
3. Neither the name of Google Inc. nor the names of its
   contributors may be used to endorse or promote products derived
   from this software without specific prior written permission.
```

clsocket adds the following clauses:

```
3. The name of the author may not be used to endorse or promote
   products derived from this software without specific prior
   written permission.

4. The name "CarrierLabs" must not be used to endorse or promote
   products derived from this software without prior written
   permission. For written permission, please contact
   mark@carrierlabs.com
```

7.3 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

7.3.1 deterioateclothes

Replaced by the new combined *deteriorate* script. Run `deteriorate --types=clothes`.

7.3.2 deterioratecorpses

Replaced by the new combined *deteriorate* script. Run `deteriorate --types=corpses`.

7.3.3 deterioratefood

Replaced by the new combined *deteriorate* script. Run `deteriorate --types=food`.

7.3.4 devel/unforbidall

Replaced by the *unforbid* script. Run `unforbid all --quiet` to match the behavior of the original `devel/unforbidall` script.

7.3.5 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`.

7.3.6 fix-armory

Allowed the military to store equipment in barracks containers. Removed because it required a binary patch to DF in order to function, and no such patch has existed since DF 0.34.11.

7.3.7 fix/build-location

The corresponding DF Bug 5991 was fixed in DF 0.40.05.

7.3.8 fix/diplomats

The corresponding DF Bug 3295 was fixed in DF 0.40.05.

7.3.9 fix/fat-dwarves

The corresponding DF Bug 5971 was fixed in DF 0.40.05.

7.3.10 fix/feeding-timers

The corresponding DF Bug 2606 was fixed in DF 0.40.12.

7.3.11 fix/merchants

Humans can now make trade agreements. This fix is no longer necessary.

7.3.12 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`.

7.3.13 gui/assign-rack

This script is no longer useful in current DF versions. The script required a binpatch <binpatches/needs-patch>`, which has not been available since DF 0.34.11.

7.3.14 gui/hack-wish

Replaced by *gui/create-item*.

7.3.15 gui/no-dfhack-init

Tool that warned the user when the `dfhack.init` file did not exist. Now that `dfhack.init` is autogenerated in `dfhack-config/init`, this warning is no longer necessary.

7.3.16 resume

Allowed you to resume suspended jobs and displayed an overlay indicating suspended building construction jobs. Replaced by *unsuspend* script.

7.3.17 warn-stuck-trees

The corresponding DF Bug 9252 was fixed in DF 0.44.01.

7.4 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

7.4.1 DFHack 0.47.05-r8

New Plugins

- *overlay*: plugin is transformed from a single line of text that runs *gui/launcher* on click to a fully-featured overlay injection framework. It now houses a popup menu for keybindings relevant to the current DF screen, all the widgets previously provided by *dwarfmonitor* (e.g. the current date and number of happy/unhappy dwarves), the overlay that highlights suspended buildings when you pause, and others. See *DFHack overlay dev guide* for details.

New Scripts

- *gui/overlay*: configuration interface for the DFHack overlays and overlay widgets. includes a click-and-drag interface for repositioning widgets!

Fixes

- Core: ensure `foo.init` always runs before `foo.*.init` (e.g. `dfhack.init` should always run before `dfhack.something.init`)
- *autofarm*: flush output so status text is visible immediately after running the command
- *autolabor*, *autohauler*: properly handle jobs 241, 242, and 243
- ***automaterial*:**
  - fix the cursor jumping up a z level when clicking quickly after box select
  - fix rendering errors with box boundary markers
- *buildingplan*: fix crash when canceling out of placement mode for a building with planning mode enabled and subsequently attempting to place a building that does not have planning mode enabled and that has no pertinent materials available
- *dwarf-op*: fixed error when matching dwarves by name
- *gui/create-item*: prevent materials list filter from intercepting sublist hotkeys
- *gui/gm-unit*: fixed behavior of `+` and `-` to adjust skill values instead of populating the search field
- *hotkeys*: correctly detect hotkeys bound to number keys, F11, and F12
- *labormanager*: associate quern construction with the correct labor
- *mousequery*: fix the cursor jumping up z levels sometimes when using TWBT
- *tiletypes*: no longer resets dig priority to the default when updating other properties of a tile
- ***warn-stealers*:**
  - register callback with correct event name so that units entering the map are detected
  - announce thieving creatures that spawn already revealed
  - cache unit IDs instead of unit objects to avoid referencing stale pointers
- *workorder*: fix interpretation of json-specified orders that set the `item_type` field
- **`EventManager`:**
  - fix a segmentation fault with the `REPORT` event
  - fix the `JOB_STARTED` event only sending events to the first handler listed instead of all registered handlers

Misc Improvements

- **UX:**
  - List widgets now have mouse-interactive scrollbars
  - You can now hold down the mouse button on a scrollbar to make it scroll multiple times.
  - You can now drag the scrollbar up and down to scroll to a specific spot
- *autolabor*, *autohauler*: refactored to use DFHack's messaging system for info/debug/trace messages
- ***blueprint*:**
  - new `--smooth` option for recording all smoothed floors and walls instead of just the ones that require smoothing for later carving
  - record built constructions in blueprints
  - record stockpile/building/zone names in blueprints
  - record room sizes in blueprints
  - generate meta blueprints to reduce the number of blueprints you have to apply
  - support splitting the output file into phases grouped by when they can be applied
  - when splitting output files, number them so they sort into the order you should apply them in
- *dig*: new `-z` option for digtype to restrict designations to the current z-level and down
- *dwarfmonitor*: widgets have been ported to the overlay framework and can be enabled and configured via the *gui/overlay* UI
- *gui/blueprint*: support new blueprint phases and options
- *gui/cp437-table*: new global keybinding for the clickable on-screen keyboard for players with keyboard layouts that prevent them from using certain keys: Ctrl-Shift-K
- *gui/create-item*: restrict materials to those normally allowed by the game by default, introduce new `--unrestricted` option for full freedom in choosing materials
- *gui/launcher*: show help for commands that start with ':' (like `:lua`)
- *gui/quantum*: add option to allow corpses and refuse in your quantum stockpile
- ***hotkeys*:**
  - hotkey screen has been transformed into an interactive *overlay* widget that you can bring up by moving the mouse cursor over the hotspot (in the upper left corner of the screen by default). Enable/disable/reposition the hotspot in the *gui/overlay* UI. Even if the hotspot is disabled, the menu can be brought up at any time with the Ctrl-Shift-C hotkey.
  - now supports printing active hotkeys to the console with `hotkeys list`
- ***ls*:**
  - indent tag listings and wrap them in the rightmost column for better readability
  - new `--exclude` option for hiding matched scripts from the output. this can be especially useful for modders who don't want their mod scripts to be included in `ls` output.
- *modtools/create-unit*: better unit naming, more argument checks, assign nemesis save data for units without civilization so they can be properly saved when offloaded
- *orders*: replace shell craft orders in the standard orders list you get with `orders import library/basic` with orders for shell leggings. They have a slightly higher trade price. Also, "shleggings" is just hilarious.

- *Quickfort blueprint library*: improved layout of marksdwarf barracks in the example Dreamfort blueprints
- ***spectate*:**
  - new `auto-unpause` option for auto-dismissal of announcement pause events (e.g. sieges).
  - new `auto-disengage` option for auto-disengagement of plugin through player interaction whilst unpaused.
  - new `tick-threshold` option for specifying the maximum time to follow the same dwarf
  - new `animals` option for sometimes following animals
  - new `hostiles` option for sometimes following hostiles
  - new `visiting` option for sometimes following visiting merchants, diplomats or plain visitors
  - added persistent configuration of the plugin settings
- *unsuspend*: new *overlay* for displaying status of suspended buildings (functionality migrated from removed *resume* plugin)

Removed

- *gui/create-item*: removed `--restricted` option. it is now the default behavior
- *resume*: functionality (including suspended building overlay) has moved to *unsuspend*

API

- Constructions module: added `insert()` to insert constructions into the game's sorted list.
- MiscUtils: added the following string transformation functions (refactored from `uicommon.h`): `int_to_string`, `ltrim`, `rtrim`, and `trim`; added `string_to_int`
- **Units module:**
  - added new predicates for:
  - `isUnitInBox()`
  - `isAnimal()`
  - `isVisiting()` any visiting unit (diplomat, merchant, visitor)
  - `isVisitor()` ie. not merchants or diplomats
  - `isInvader()`
  - `isDemon()` returns true for unique/regular demons
  - `isTitan()`
  - `isMegabeast()`
  - `isGreatDanger()` returns true if unit is a demon, titan, or megabeast
  - `isSemiMegabeast()`
  - `isNightCreature()`
  - `isDanger()` returns true if is a 'GreatDanger', semi-megabeast, night creature, undead, or invader
  - modified predicates:
  - `isUndead()` now optionally ignores vampires instead of always ignoring vampires

    - `isCitizen()` now optionally ignores insane citizens instead of always ignoring insane citizens

- `Gui::anywhere_hotkey`: for plugin commands bound to keybindings that can be invoked on any screen
- `Gui::autoDFAnnouncement`, `Gui::pauseRecenter`: added functionality reverse-engineered from announcement code
- `Gui::revealInDwarfmodeMap`: Now enforce valid view bounds when pos invalid, add variant accepting x, y, z
- `Lua::Push()`: now handles maps with otherwise supported keys and values
- `Lua::PushInterfaceKeys()`: transforms viewscreen `feed()` keys into something that can be interpreted by lua-based widgets

Internals

- Constructions module: `findAtTile` now uses a binary search intead of a linear search
- MSVC warning level upped to /W3, and /WX added to make warnings cause compilations to fail.

Lua

- Lua mouse events now conform to documented behavior in *DFHack Lua API Reference* – `_MOUSE_L_DOWN` will be sent exactly once per mouse click and `_MOUSE_L` will be sent repeatedly as long as the button is held down. Similarly for right mouse button events.
- `dfhack.constructions.findAtTile()`: exposed preexisting function to Lua.
- `dfhack.constructions.insert()`: exposed new function to Lua.
- `gui.Screen.show()`: now returns `self` as a convenience
- `gui.View.getMousePos()` now takes an optional `ViewRect` parameter in case the caller wants to get the mouse pos relative to a rect that is not the frame_body (such as the frame_rect that includes the frame itself)
- `widgets.EditField`: now allows other widgets to process characters that the `on_char` callback rejects.
- `widgets.FilteredList`: now provides a useful default search key for list items made up of text tokens instead of plain text
- `widgets.HotkeyLabel`: now ignores mouse clicks when `on_activate` is not defined
- **`widgets.List`:**
    - new `getIdxUnderMouse()` function for detecting the list index under the active mouse cursor. this allows for “selection follows mouse” behavior
    - shift-clicking now triggers the `submit2` attribute function if it is defined
- `widgets.Panel`: new `frame_style` and `frame_title` attributes for drawing frames around groups of widgets
- `widgets.ResizingPanel`: now accounts for frame inset when calculating frame size
- `widgets.Scrollbar`: new scrollbar widget that can be paired with an associated scrollable widget. Integrated with `widgets.Label` and `widgets.List`.

Structures

- `general_refst`: type virtual union member for `ITEM_GENERAL`
- `historical_figure_info.T_reputation.unk_2c`: identify `year` + `year_ticks`
- `itemst`: identify two vmethods related to adding thread improvements to items made of cloth, and label several previously unknown return types
- `proj_magicst`: correct structure fields (to match 40d)
- `unit_action_type_group`: added enum and tagged `unit_action_type` entries with its groups for DFHack's new action timer API.
- `world`: identify type of a vector (still not known what it's for, but it's definitely an item vector)

Documentation

- *DFHack overlay dev guide*: documentation and guide for injecting functionality into DF viewscreens from Lua scripts and creating interactive overlay widgets
- `dfhack.gui.revealInDwarfmodeMap`: document `center` bool for Lua API

7.4.2 DFHack 0.47.05-r7

New Plugins

- *autobutcher*: split off from *zone* into its own plugin. Note that to enable, the command has changed from `autobutcher start` to `enable autobutcher`.
- *autonestbox*: split off from *zone* into its own plugin. Note that to enable, the command has changed from `autonestbox start` to `enable autonestbox`.
- *overlay*: display a "DFHack" button in the lower left corner that you can click to start the new GUI command launcher. The *dwarfmonitor* weather display had to be moved to make room for the button. If you are seeing the weather indicator rendered over the overlay button, please remove the `dfhack-config/dwarfmonitor.json` file to fix the weather indicator display offset.

New Scripts

- *gui/kitchen-info*: adds more info to the Kitchen screen
- *gui/launcher*: in-game command launcher with autocomplete, history, and context-sensitive help
- *gui/workorder-details*: adjusts work orders' input item, material, traits
- *max-wave*: dynamically limit the next immigration wave, can be set to repeat
- *pop-control*: persistent per fortress population cap, *hermit*, and *max-wave* management
- *warn-stealers*: warn when creatures that may steal your food, drinks, or items become visible

New Internal Commands

- *tags*: new built-in command to list the tool category tags and their definitions. tags associated with each tool are visible in the tool help and in the output of *ls*.

Fixes

- *autochop*: designate largest trees for chopping first, instead of the smallest
- *devel/query*: fixed error when –tile is specified
- *dig-now*: Fix direction of smoothed walls when adjacent to a door or floodgate
- *dwarf-op*: fixed error when applying the Miner job to dwarves
- *emigration*: fix emigrant logic so unhappy dwarves leave as designed
- *gui/gm-unit*: allow + and - to adjust skill values as intended instead of letting the filter intercept the characters
- *gui/unit-info-viewer*: fix logic for displaying undead creature names
- *gui/workflow*: restore functionality to the add/remove/order hotkeys on the workflow status screen
- *modtools/moddable-gods*: fixed an error when assigning spheres
- *quickfort*: *Dreamfort* blueprint set: declare the hospital zone before building the coffer; otherwise DF fails to stock the hospital with materials
- *view-item-info*: fixed a couple errors when viewing items without materials
- `dfhack.buildings.findCivzonesAt`: no longer return duplicate civzones after loading a save with existing civzones
- `dfhack.run_script`: ensure the arguments passed to scripts are always strings. This allows other scripts to call `run_script` with numeric args and it won't break parameter parsing.
- `job.removeJob()`: ensure jobs are removed from the world list when they are canceled

Misc Improvements

- History files: `dfhack.history`, `tiletypes.history`, `lua.history`, and `liquids.history` have moved to the `dfhack-config` directory. If you'd like to keep the contents of your current history files, please move them to `dfhack-config`.
- Init scripts: `dfhack.init` and other init scripts have moved to `dfhack-config/init/`. If you have customized your `dfhack.init` file and want to keep your changes, please move the part that you have customized to the new location at `dfhack-config/init/dfhack.init`. If you do not have changes that you want to keep, do not copy anything, and the new defaults will be used automatically.
- **UX:**
  - You can now move the cursor around in DFHack text fields in `gui/` scripts (e.g. *gui/blueprint*, *gui/quickfort*, or *gui/gm-editor*). You can move the cursor by clicking where you want it to go with the mouse or using the Left/Right arrow keys. Ctrl+Left/Right will move one word at a time, and Alt+Left/Right will move to the beginning/end of the text.
  - You can now click on the hotkey hint text in many `gui/` script windows to activate the hotkey, like a button. Not all scripts have been updated to use the clickable widget yet, but you can try it in *gui/blueprint* or *gui/quickfort*.

  - Label widget scroll icons are replaced with scrollbars that represent the percentage of text on the screen and move with the position of the visible text, just like web browser scrollbars.

- ***devel/query*:**
  - inform the user when a query has been truncated due to `--maxlength` being hit.
  - increased default maxlength value from 257 to 2048
- *do-job-now*: new global keybinding for boosting the priority of the jobs associated with the selected building/work order/unit/item etc.: Alt-N
- *dwarf-op*: replaces [a b c] option lists with a,b,c option lists
- *gui/gm-unit*: don't clear the list filter when you adjust a skill value
- ***gui/quickfort*:**
  - better formatting for the generated manager orders report
  - you can now click on the map to move the blueprint anchor point to that tile instead of having to use the cursor movement keys
  - display an error message when the blueprints directory cannot be found
- *gui/workorder-details*: new keybinding on the workorder details screen: `D`
- *keybinding*: support backquote (`` ` ``) as a hotkey (and assign the hotkey to the new *gui/launcher* interface)
- *ls*: can now filter tools by substring or tag. note that dev scripts are hidden by default. pass the `--dev` option to show them.
- ***manipulator*:**
  - add a library of useful default professions
  - move professions configuration from `professions/` to `dfhack-config/professions/` to keep it together with other dfhack configuration. If you have saved professions that you would like to keep, please manually move them to the new folder.
- *orders*: added useful library of manager orders. see them with `orders list` and import them with, for example, `orders import library/basic`
- *prioritize*: new `defaults` keyword to prioritize the list of jobs that the community agrees should generally be prioritized. Run `prioritize -a defaults` to try it out in your fort!
- *prospector*: add new `--show` option to give the player control over which report sections are shown. e.g. `prospect all --show ores` will just show information on ores.
- ***quickfort*:**
  - *Dreamfort* blueprint set improvements: set traffic designations to encourage dwarves to eat cooked food instead of raw ingredients
  - library blueprints are now included by default in `quickfort list` output. Use the new `--useronly` (or just `-u`) option to filter out library bluerpints.
  - better error message when the blueprints directory cannot be found
- *seedwatch*: `seedwatch all` now adds all plants with seeds to the watchlist, not just the "basic" crops.
- `materials.ItemTraitsDialog`: added a default `on_select`-handler which toggles the traits.

Removed

- *fix/build-location*: The corresponding DF bug (5991) was fixed in DF 0.40.05
- *fix/diplomats*: DF bug 3295 fixed in 0.40.05
- *fix/fat-dwarves*: DF bug 5971 fixed in 0.40.05
- *fix/feeding-timers*: DF bug 2606 is fixed in 0.40.12
- *fix/merchants*: DF bug that prevents humans from making trade agreements has been fixed
- *gui/assign-rack*: No longer useful in current DF versions
- *gui/hack-wish*: Replaced by *gui/create-item*
- *gui/no-dfhack-init*: No longer useful since players don't have to create their own `dfhack.init` files anymore

API

- Removed "egg" ("eggy") hook support (Linux only). The only remaining method of hooking into DF is by interposing SDL calls, which has been the method used by all binary releases of DFHack.
- Removed `Engravings` module (C++-only). Access `world.engravings` directly instead.
- Removed `Notes` module (C++-only). Access `ui.waypoints.points` directly instead.
- Removed `Windows` module (C++-only) - unused.
- `Constructions` module (C++-only): removed `t_construction`, `isValid()`, `getCount()`, `getConstruction()`, and `copyConstruction()`. Access `world.constructions` directly instead.
- `Gui::getSelectedItem()`, `Gui::getAnyItem()`: added support for the artifacts screen
- `Units::teleport()`: now sets `unit.idle_area` to discourage units from walking back to their original location (or teleporting back, if using *fastdwarf*)

Lua

- Added `dfhack.screen.hideGuard()`: exposes the C++ `Screen::Hide` to Lua
- History: added `dfhack.getCommandHistory(history_id, history_filename)` and `dfhack.addCommandToHistory(history_id, history_filename, command)` so gui scripts can access a commandline history without requiring a terminal.
- `helpdb`: database and query interface for DFHack tool help text
- `tile-material`: fix the order of declarations. The `GetTileMat` function now returns the material as intended (always returned nil before). Also changed the license info, with permission of the original author.
- `utils.df_expr_to_ref()`: fixed some errors that could occur when navigating tables
- `widgets.CycleHotkeyLabel`: clicking on the widget will now cycle the options and trigger `on_change()`. This also applies to the `ToggleHotkeyLabel` subclass.
- **`widgets.EditField`:**
  - new `onsubmit2` callback attribute is called when the user hits Shift-Enter.
  - new function: `setCursor(position)` sets the input cursor.
  - new attribute: `ignore_keys` lets you ignore specified characters if you want to use them as hotkeys

- `widgets.FilteredList`: new attribute: `edit_ignore_keys` gets passed to the filter EditField as `ignore_keys`
- `widgets.HotkeyLabel`: clicking on the widget will now call `on_activate()`.
- `widgets.Label`: `scroll` function now interprets the keywords `+page`, `-page`, `+halfpage`, and `-halfpage` in addition to simple positive and negative numbers.

Structures

- Eliminate all “anon_X” names from structure fields
- `army`: change `squads` vector type to `world_site_inhabitant`, identify `min_smell_trigger``+``max_odor_level``+``max_low_light_vision``+``sense_creature_classes`
- `cave_column_rectangle`: identify coordinates
- `cave_column`: identify Z coordinates
- `embark_profile`: identify reclaim fields, add missing pet_count vector
- `entity_population`: identify `layer_id`
- `feature`: identify “shiftCoords” vmethod, `irritation_level` and `irritation_attacks` fields
- `flow_guide`: identify “shiftCoords” vmethod
- `general_refst`: name parameters on `getLocation` and `setLocation` vmethods
- `general_ref_locationst`: name member fields
- `historical_entity`: confirm `hostility_level` and `siege_tier`
- `item`: identify method `notifyCreatedMasterwork` that is called when a masterwork is created.
- `language_name_type`: identify `ElfTree` and `SymbolArtifice` thru `SymbolFood`
- `misc_trait_type`: update auto-decrement markers, remove obsolete references
- `timed_event`: identify `layer_id`
- `ui_advmode`: identify several fields as containing coordinates
- `ui_build_selector`: identify `cur_walk_tag` and `min_weight_races``+``max_weight_races`
- `ui`: identify actual contents of `unk5b88` field, identify infiltrator references
- `unitst`: identify `histeventcol_id` field inside status2
- `viewscreen_barterst`: name member fields
- `viewscreen_tradegoodsst`: rename trade_reply `OffendedAnimal``+``OffendedAnimalAlt` to `OffendedBoth``+``OffendedAnimal`
- `world_site_inhabitant`: rename `outcast_id` and `founder_outcast_entity_id`, identify `interaction_id` and `interaction_effect_idx`

Documentation

- Added *DFHack modding guide*
- Group DFHack tools by *tag* so similar tools are grouped and easy to find
- Update all DFHack tool documentation (300+ pages) with standard syntax formatting, usage examples, and overall clarified text.

7.4.3 DFHack 0.47.05-r6

New Scripts

- *assign-minecarts*: automatically assign minecarts to hauling routes that don't have one
- *deteriorate*: combines, replaces, and extends previous *deterioratеclothes*, *deterioratecorpses*, and *deteriorate-food* scripts.
- *gui/petitions*: shows petitions. now you can see which guildhall/temple you agreed to build!
- *gui/quantum*: point-and-click tool for creating quantum stockpiles
- *gui/quickfort*: shows blueprint previews on the live map so you can apply them interactively
- *modtools/fire-rate*: allows modders to adjust the rate of fire for ranged attacks

Fixes

- *build-now*: walls built above other walls can now be deconstructed like regularly-built walls
- ***eventful*:**
  - fix `eventful.registerReaction` to correctly pass `call_native` argument thus allowing canceling vanilla item creation. Updated related documentation.
  - renamed NEW_UNIT_ACTIVE event to UNIT_NEW_ACTIVE to match the `EventManager` event name
  - fixed UNIT_NEW_ACTIVE event firing too often
- *gui/dfstatus*: no longer count items owned by traders
- *gui/unit-info-viewer*: fix calculation/labeling of unit size
- `job.removeJob()`: fixes regression in DFHack 0.47.05-r5 where items/buildings associated with the job were not getting disassociated when the job is removed. Now *build-now* can build buildings and *gui/mass-remove* can cancel building deconstruction again
- `widgets.CycleHotkeyLabel`: allow initial option values to be specified as an index instead of an option value

Misc Improvements

- *build-now*: buildings that were just designated with *buildingplan* are now built immediately (as long as there are items available to build the buildings with) instead of being skipped until buildingplan gets around to doing its regular scan
- *caravan*: new `unload` command, fixes endless unloading at the depot by reconnecting merchant pack animals that were disconnected from their owners
- ***confirm***:
  - added a confirmation dialog for removing manager orders
  - allow players to pause the confirmation dialog until they exit the current screen
- *deteriorate*: new `now` command immediately deteriorates items of the specified types
- ***DFHack config file examples***:
  - refine food preparation orders so meal types are chosen intelligently according to the amount of meals that exist and the number of aviailable items to cook with
  - reduce required stock of dye for "Dye cloth" orders
  - fix material conditions for making jugs and pots
  - make wooden jugs by default to differentiate them from other stone tools. this allows players to more easily select jugs out with a properly-configured stockpile (i.e. the new `woodentools` alias)
- *list-agreements*: now displays translated guild names, worshipped deities, petition age, and race-appropriate professions (e.g. "Craftsdwarf" instead of "Craftsman")
- ***Quickfort keystroke alias reference***:
  - new aliases: `forbidsearch`, `permitsearch`, and `togglesearch` use the *search* plugin to alter the settings for a filtered list of item types when configuring stockpiles
  - new aliases: `stonetools` and `woodentools`. the `jugs` alias is deprecated. please use `stonetools` instead, which is the same as the old `jugs` alias.
  - new aliases: `usablehair`, `permitusablehair`, and `forbidusablehair` alter settings for the types of hair/wool that can be made into cloth: sheep, llama, alpaca, and troll. The `craftrefuse` aliases have been altered to use this alias as well.
  - new aliases: `forbidthread`, `permitthread`, `forbidadamantinethread`, `permitadamantinethread`, `forbidcloth`, `permitcloth`, `forbidadamantinecloth`, and `permitadamantinecloth` give you more control how adamantine-derived items are stored
- ***quickfort***:
  - *Dreamfort* blueprint set improvements: automatically create tavern, library, and temple locations (restricted to residents only by default), automatically associate the rented rooms with the tavern
  - *Dreamfort* blueprint set improvements: new design for the services level, including were-bitten hospital recovery rooms and an appropriately-themed interrogation room next to the jail! Also fits better in a 1x1 embark for minimalist players.
- *workorder*: a manager is no longer required for orders to be created (matching bevavior in the game itself)

Removed

- *deterioratеclothes*: please use `deteriorate --types=clothes` instead
- *deterioratecorpses*: please use `deteriorate --types=corpses` instead
- *deterioratefood*: please use `deteriorate --types=food` instead
- *devel/unforbidall*: please use *unforbid* instead. You can silence the output with `unforbid all --quiet`

API

- `word_wrap`: argument `bool collapse_whitespace` converted to enum `word_wrap_whitespace_mode mode`, with valid modes `WSMODE_KEEP_ALL`, `WSMODE_COLLAPSE_ALL`, and `WSMODE_TRIM_LEADING`.

Lua

- `gui.View`: all `View` subclasses (including all `Widgets`) can now acquire keyboard focus with the new `View:setFocus()` function. See docs for details.
- `materials.ItemTraitsDialog`: new dialog to edit item traits (where "item" is part of a job or work order or similar). The list of traits is the same as in vanilla work order conditions "`t` change traits".
- **`widgets.EditField`:**
  - the `key_sep` string is now configurable
  - can now display an optional string label in addition to the activation key
  - views that have an `EditField` subview no longer need to manually manage the `EditField` activation state and input routing. This is now handled automatically by the new `gui.View` keyboard focus subsystem.
- `widgets.HotkeyLabel`: the `key_sep` string is now configurable

Structures

- `art_image_elementst`: identify vmethod `markDiscovered` and second parameter for `getName2`
- `art_image_propertyst`: identify parameters for `getName`
- `building_handler`: fix vmethod `get_machine_hookup_list` parameters
- `vermin`: identify `category` field as new enum
- `world.unk_26a9a8`: rename to `allow_announcements`

7.4.4 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 config file examples***:
  - 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()`
- `Job::removeJob()`: use the job cancel vmethod graciously provided by The Toady One in place of a synthetic method derived from reverse engineering

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, `Panel`s 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 extensions*: document DFHack string extensions (`startswith()`, `endswith()`, `split()`, `trim()`, `wrap()`, and `escape_pattern()`)
- *Quickfort blueprint creation 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)`

7.4.5 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*, *tiletypes*: 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
- *Quickfort blueprint library*: 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 config file examples*: 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

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`

7.4.6 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. *Data 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*
- *Data 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*, *tiletypes*: 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 config file examples*: documentation for all of *Dreamfort*'s supporting files (useful for all forts, not just Dreamfort!)
- *Quickfort blueprint library*: updated dreamfort documentation and added screenshots

7.4.7 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*

7.4.8 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
- *prospector*: 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*

7.4.9 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

7.4.10 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 reference*: alias syntax and alias standard library documentation for *quickfort* blueprints
- *Quickfort blueprint library*: overview of the quickfort blueprint library

7.4.11 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 creation 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

7.4.12 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

7.4.13 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
- *unretire-anyone*: added `-dead` argument to revive and enable selection of a dead historical figure to use as an adventurer in adv mode
- `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

7.4.14 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*)

7.4.15 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
- *prospector*: (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

7.4.16 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. *cleaners*
- *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_civlistst`)
- `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_civlistst`:**
  - 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_reportlistst`: 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

7.4.17 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*

- ***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
- `item-descriptions`: fixed several grammatical errors

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)

7.4.18 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
- *prospector*: 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)

7.4.19 DFHack 0.44.09-r1

Fixes

- Fixed some CMake warnings (CMP0022)
- Support for building on Ubuntu 18.04
- *dig*: 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 renaming:**
  - `world.reactions` to `world.reactions.reactions`
  - `world.reaction_categories` to `world.reactions.reaction_categories`

7.4.20 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

7.4.21 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_reportlistst`: 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`

7.4.22 Older Changelogs

Are kept in a separate file: History

INDEX OF DFHACK TOOLS

"ADVENTURE" TAG INDEX - TOOLS THAT ARE USEFUL WHILE IN ADVENTURE MODE. NOTE THAT SOME TOOLS ONLY TAGGED WITH "FORT" MIGHT ALSO WORK IN ADVENTURE MODE, BUT NOT ALWAYS IN EXPECTED WAYS. FEEL FREE TO EXPERIMENT, THOUGH!

696"adventure" tag index - Tools that are useful while in adventure mode. Note that some tools only tagged with "fort" might also work in adventure mode, but not always in expected ways. Feel free to experiment, though!

"DFHACK" TAG INDEX - TOOLS THAT YOU USE TO RUN DFHACK COMMANDS OR INTERACT WITH THE DFHACK LIBRARY. THIS TAG ALSO INCLUDES TOOLS THAT HELP YOU MANAGE THE DF GAME ITSELF (E.G. SETTINGS, SAVING, ETC.)

"EMBARK" TAG INDEX - TOOLS THAT ARE USEFUL WHILE ON THE FORT EMBARK SCREEN OR WHILE CREATING AN ADVENTURER.

"FORT" TAG INDEX - TOOLS THAT ARE USEFUL WHILE IN FORT MODE.

"LEGENDS" TAG INDEX - TOOLS THAT ARE USEFUL WHILE IN LEGENDS MODE.

"ARMOK" TAG INDEX - TOOLS THAT GIVE YOU COMPLETE CONTROL OVER AN ASPECT OF THE GAME OR PROVIDE ACCESS TO INFORMATION THAT THE GAME INTENTIONALLY KEEPS HIDDEN.

"AUTO" TAG INDEX - TOOLS THAT RUN IN THE BACKGROUND AND AUTOMATICALLY MANAGE ROUTINE, TOILSOME ASPECTS OF YOUR FORTRESS.

"BUGFIX" TAG INDEX - TOOLS THAT FIX SPECIFIC BUGS, EITHER PERMANENTLY OR ON-DEMAND.

"DESIGN" TAG INDEX - TOOLS THAT HELP YOU DESIGN YOUR FORT.

"DEV" TAG INDEX - TOOLS THAT ARE USEFUL WHEN DEVELOPING SCRIPTS OR MODS.

"FPS" TAG INDEX - TOOLS THAT HELP YOU MANAGE FPS DROP.

"GAMEPLAY" TAG INDEX - TOOLS THAT INTRODUCE NEW GAMEPLAY ELEMENTS.

"INSPECTION" TAG INDEX - TOOLS THAT LET YOU VIEW INFORMATION THAT IS OTHERWISE DIFFICULT TO FIND.

"PRODUCTIVITY" TAG INDEX - TOOLS THAT HELP YOU DO THINGS THAT YOU COULD DO MANUALLY, BUT USING THE TOOL IS BETTER AND FASTER.

"ANIMALS" TAG INDEX - TOOLS THAT INTERACT WITH ANIMALS.

"BUILDINGS" TAG INDEX - TOOLS THAT INTERACT WITH BUILDINGS AND FURNITURE.

"GRAPHICS" TAG INDEX - TOOLS THAT INTERACT WITH GAME GRAPHICS.

"INTERFACE" TAG INDEX - TOOLS THAT INTERACT WITH OR EXTEND THE DF USER INTERFACE.

"ITEMS" TAG INDEX - TOOLS THAT INTERACT WITH IN-GAME ITEMS.

"JOBS" TAG INDEX - TOOLS THAT INTERACT WITH JOBS.

"LABORS" TAG INDEX - TOOLS THAT DEAL WITH LABOR ASSIGNMENT.

"MAP" TAG INDEX - TOOLS THAT INTERACT WITH THE GAME MAP.

"MILITARY" TAG INDEX - TOOLS THAT INTERACT WITH THE MILITARY.

"PLANTS" TAG INDEX - TOOLS THAT INTERACT WITH TREES, SHRUBS, AND CROPS.

"STOCKPILES" TAG INDEX - TOOLS THAT INTERACT WITH STOCKPILES.

"UNITS" TAG INDEX - TOOLS THAT INTERACT WITH UNITS.

"WORKORDERS" TAG INDEX - TOOLS THAT INTERACT WITH WORKORDERS.