DFHack Documentation

Release 50.13-r2

The DFHack Team

2024-05-09

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*
- *Quickstart guide*
- *Getting help*
- **Source code**
  (**important:** read *Building 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 hosted on GitHub, and can be downloaded from the releases page – see *Installing* for installation instructions. This is also where the DFHack bug tracker is hosted. If you would like to download the DFHack documentation for offline viewing, you can do so by clicking the expansion panel in the lower right corner of our online documentation and selecting your desired format from the "Downloads" section.

New releases are announced in the DF subreddit, the DFHack Discord, and the Bay12 forums thread. Discussion and questions are also welcome in each of these venues.

For users, DFHack provides a significant suite of bugfixes and interface enhancements by default, and more features can be enabled as desired. There are also many tools (such as *autofarm*) which automate aspects of gameplay many players find toilsome. 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. 3rd party DFHack scripts can be distributed *in mods* via the DF Steam Workshop or on the forums.

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* for details about installing DFHack.

Once DFHack is installed, it extends DF with a console that can be used to run commands. The in-game version of this console is called *gui/launcher*, and you can bring it up at any time by hitting the backtick (`) key (on most keyboards this is the same as the tilde (~) key). There are also external consoles you can open in a separate window. On Windows, you can show this console with the *show* command. On Linux and macOS, you will need to run the `dfhack` script from a terminal, and that terminal will be used as 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. The first command you should run is likely *gui/control-panel* so you can set up which tools you would like to enable now and which tools you want automatically started for new games.

Another way to interact with DFHack is to set in-game *keybindings* to run commands in response to a hotkey. If you have specific commands that you run frequently and that don't already have default keybindings, this can be a better option than adding the command to the *gui/quickcmd* list.

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 surround the added text or clickable buttons in red square brackets.

For a more thorough introduction and guide through DFHack's capabilities, please see the *Quickstart guide*.

1.2 Getting help

DFHack has several ways to get help online, including:

- The DFHack Discord server
- **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
- The /r/dwarffortress questions thread on Reddit

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. Log output from `stderr.log` in the DF folder can often help point to the cause of issues.

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

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

CHAPTER

TWO

QUICKSTART GUIDE

Welcome to DFHack! This guide will help get you oriented with the DFHack system and teach you how to find and use the tools productively. If you're reading this in the in-game *quickstart-guide* reader, hit the right arrow key or click on the hotkey hint in the lower right corner of the window to go to the next page.

2.1 What is DFHack?

DFHack is an add-on for Dwarf Fortress that enables mods and tools to significantly extend the game. The default DFHack distribution contains a wide variety of these mods and tools, including bugfixes, interface improvements, automation agents, design blueprints, modding building blocks, and more. Third-party tools (e.g. mods downloaded from Steam Workshop or the forums) can also seamlessly integrate with the DFHack framework and extend the game far beyond what can be done by just modding the raws.

DFHack's mission is to provide tools and interfaces for players and modders to:

- expand the bounds of what is possible in Dwarf Fortress
- reduce the impact of game bugs
- give the player more agency and control over the game
- provide alternatives to toilsome or frustrating aspects of gameplay
- **make the game more fun**

2.2 What can I do with DFHack tools?

DFHack has been around for a long time – almost as long as Dwarf Fortress itself. Many of the game's rough edges have been smoothed with DFHack tools. Here are some common tasks people use DFHack tools to accomplish:

- Automatically chop trees when log stocks are low
- Mark all damaged items for trade in a single click
- Copy and paste fort layouts
- Import and export lists of manager orders
- Clean contaminants from map squares that dwarves can't reach
- Automatically butcher excess livestock so you don't become overrun with animals
- Promote time-sensitive job types (e.g. food hauling) so they are done expediently
- Quickly scan the map for visible ores of specific types so you can focus your mining efforts

Some tools are one-shot commands. For example, you can run *unforbid all* to claim all (reachable) items on the map after a messy siege.

Other tools must be *enabled* once and then they will run in the background. For example, once enabled, *seedwatch* will start monitoring your stocks of seeds and prevent your chefs from cooking seeds that you need for planting. Tools that are enabled in the context of a fort will save their state with that fort, and they will remember that they are enabled the next time you load your save. You can see which tools you have enabled and toggle their states in *gui/control-panel*.

A third class of tools adds information to the screen or provides new integrated functionality via the DFHack *overlay* framework. For example, the *sort* tool adds widgets to the squad member selection screen that allow you to sort and filter the list of military candidates. You don't have to run any command to get the benefits of the tool, it appears automatically when you're on the relevant screen.

2.3 How can I figure out which commands to run?

There are several ways to scan DFHack tools and find the ones you need right now.

The first place to check is the DFHack logo menu. It's in the upper left corner of the screen by default, though you can move it anywhere you want with the *gui/overlay* configuration UI.

When you click on the logo (or hit the Ctrl-Shift-C keyboard shortcut), a short list of popular, relevant DFHack tools comes up. These are the tools that have been assigned hotkeys that are active in the current context. For example, when you're looking at a fort map, the list will contain fortress design tools like *gui/quickfort* and *gui/design*. You can click on the tools in the list, or note the hotkeys listed next to them and maybe use them to launch the tool next time without even opening the logo menu.

The second place to check is the DFHack control panel: *gui/control-panel*. It will give you an overview of which tools are currently enabled, and will allow you to toggle tools on or off, see help text for them, or launch their dedicated configuration UIs. You can open the control panel from anywhere with the Ctrl-Shift-E hotkey or by selecting it from the logo menu list.

In the control panel, you can also select which tools you'd like to be automatically enabled and popular commands you'd like to run when you start a new fort. On the "Preferences" tab, there are settings you can change, like the aforementioned "mortal mode" or whether you want DFHack windows to pause the game when they come up.

Finally, you can explore the full extent of the DFHack catalog in *gui/launcher*, which is always listed first in the DFHack logo menu list. You can bring up the launcher by tapping the backtick key (`) or hitting Ctrl-Shift-D. In the launcher, you can quickly autocomplete any command name by selecting it in the list on the right side of the window. You can filter the list by command tag, for example, you can see only productivity tools by setting the "productivity" tag to "include" in the filter panel. Commands are ordered by how often you run them, so your favorite commands will always be on top. You can also pull full commandlines out of your history with Alt-S or by clicking on the "history search" button.

Once you have typed (or autocompleted, or searched for) a command, other commands related to the one you have selected will appear in the right-hand panel. Scanning through that list is a great way to learn about new tools that you might find useful.

The bottom panel will show the full help text for the command you are running, allowing you to refer to the usage documentation and examples when you are typing your command. After you run a command, the bottom panel switches to command output mode, but you can get back to the help text by hitting Ctrl-T or clicking on the `Help` tab.

2.4 What if I don't want to be tempted by god-mode tools?

DFHack can give you god-like powers over the game. Sometimes, this is necessary to recover from game-breaking bugs. Sometimes, this is desirable so players can create specific role playing environments and situations. Sometimes, this is just fun :)

But sometimes the knowledge that you can just "Armok" your way out of trouble detracts from the game experience. If this is the way you feel, you can hide DFHack's god-mode tools – they all have the tag "armok" – from the game. Open *gui/control-panel* and go to the "Preferences" tab. Enable "Mortal mode" to hide all "armok" tools. You can still access them in *gui/launcher* if you type their names in full, but they won't show up in autocomplete lists, in the output of *ls*, or anywhere else. Any global hotkeys that run "armok" tools will be disabled.

2.5 How do DFHack in-game windows work?

Many DFHack tools have graphical interfaces that appear in-game. You can tell which windows belong to DFHack tools because they will have the word "DFHack" printed across their bottom frame edge. DFHack provides an advanced windowing system that gives the player a lot of control over where the windows appear and whether they capture keyboard and mouse input.

The DFHack windowing system allows multiple overlapping windows to be active at once. The one with the highlighted title bar has focus and will receive anything you type at the keyboard. Hit Esc or right click to close the window or cancel the current action. You can click anywhere on the screen that is not a DFHack window to unfocus the window and let it just sit in the background. It won't respond to key presses or mouse clicks until you click on it again to give it focus. If no DFHack windows are focused, you can right click directly on a window to close it without left clicking to focus it first.

DFHack windows are draggable from the title bar or from anywhere on the window that doesn't have a mouse-clickable widget on it. Many are resizable as well (if the tool window has components that can reasonably be resized).

You can generally use DFHack tools without interrupting the game. That is, if the game is unpaused, it can continue to run while a DFHack window is open. If configured to do so in *gui/control-panel*, tools will initially pause the game to let you focus on the task at hand, but you can unpause like normal if you want. You can also interact with the map, scrolling it with the keyboard or mouse and selecting units, buildings, and items. Some tools, like *gui/blueprint*, will intercept all mouse clicks to allow you to select regions of the map. When these tools have focus, you will not be able to use the mouse to interact with map elements or pause/unpause the game. Therefore, these tools will pause the game when they open, regardless of your settings in *gui/control-panel*. You can still unpause with the keyboard (spacebar by default), though.

2.6 Where do I go next?

To recap:

You can get to popular, relevant tools for the current context by clicking on the DFHack logo or by hitting Ctrl-Shift-C.

You can enable DFHack tools and configure settings with *gui/control-panel*, which you can open from the DFHack logo or access directly with the Ctrl-Shift-E hotkey.

You can get to the launcher and its integrated autocomplete, history search, and help text by hitting backtick (`) or Ctrl-Shift-D, or, of course, by running it from the logo menu list.

With those three interfaces, you have the complete DFHack tool suite at your fingertips. So what to run first? Here are a few examples to get you started.

First, let's import some useful manager orders to keep your fort stocked with basic necessities. Run `orders import library/basic`. If you go to your manager orders screen, you can see all the orders that have been created for you. Note that you could have imported the orders directly from this screen as well, using the DFHack *overlay* widget at the bottom of the manager orders panel.

Next, try setting up *autochop* to automatically designate trees for chopping when you get low on usable logs. Run *gui/control-panel* and enable `autochop` in the `Automation -> Enabled` tab. Click on the button to the left of the name or hit Enter to enable it. You can then click on the configure button (the gear icon) to launch *gui/autochop* if you'd like to customize its settings (the defaults are usually fine). If you have the extra screen space, you can go ahead and set the *gui/autochop* window to minimal mode (click on the button near the upper right corner of the window or hit Alt-M) and click on the map so the window loses keyboard focus. As you play the game, you can glance at the live status panel to check on your stocks of wood.

Finally, let's do some fort design copy-pasting. Go to some bedrooms that you have set up in your fort. Run *gui/blueprint*, set a name for your blueprint by clicking on the name field (or hitting the 'n' hotkey). Type "rooms" (or whatever) and hit Enter to set. Then draw a box around the target area by clicking with the mouse. When you select the second corner, the blueprint will be saved to your `dfhack-config/blueprints` subfolder.

Now open up *gui/quickfort* (the hotkey is Ctrl-Shift-Q). You can search for the blueprint you just created by typing its name, but it should be up near the top already. If you copied a dug-out area with furniture in it, you will see two blueprints with the labels "/dig" and "/build". Click on the "/dig" blueprint or select it with the keyboard arrow keys and hit Enter. You can rotate or flip the blueprint around if you need to with the transform hotkeys. You'll see a preview of where the blueprint will be applied as you move the mouse cursor around the map. Red outlines mean that the blueprint may fail to fully apply at that location, so be sure to choose a spot where all the preview tiles are shown with green diamonds. Click the mouse or hit Enter to apply the blueprint and designate the tiles for digging. Your dwarves will come and dig it out as if you had designated the tiles yourself.

Once the area is dug out, run *gui/quickfort* again and select the "/build" blueprint this time. Hit `o` to generate manager orders for the required furniture. Click to apply the blueprint in the dug-out area, and your furniture will be designated. It's just that easy! Note that *quickfort* uses *buildingplan* to place buildings, so you don't even need to have the relevant furniture or building materials in stock yet. The planned furniture/buildings will get built whenever you are able to produce the building materials.

There are many, many more tools to explore. Poke around or ask other player for advice. Have fun, and dig deep!

CHAPTER

THREE

INSTALLING

3.1 Requirements

DFHack supports all operating systems and platforms that Dwarf Fortress itself supports, which at the moment is the 64-bit versions of Windows and Linux. The Windows build of DFHack also works well under `wine` for platforms that can't run a native version. When running via `wine`, use the following commandline:

```
wine64 explorer Dwarf\ Fortress.exe
```

DFHack releases generally only support the version of Dwarf Fortress that they are named after. For example, DFHack 50.05 only supported DF 50.05. DFHack releases *never* support newer versions of DF – 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 use the latest versions of both DF and DFHack.

3.2 Downloading DFHack

Stable builds of DFHack are available on Steam or from our GitHub. Either location will give you exactly the same package.

On Steam, note that DFHack is a separate app, not a DF Steam Workshop mod. You can run DF with DFHack by launching either the DFHack app or the original Dwarf Fortress app.

If you download from GitHub, 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-50.07-r1-Windows-64bit.zip` supports 64-bit DF on Windows

> **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, please see *Building DFHack* for instructions.)

3.2.1 Beta releases

In between stable releases, we may create beta releases to test new features. These are available via the `beta` release channel on Steam or from our regular Github page as a pre-release tagged with a "beta" or "rc" ("release candidate") suffix.

3.2.2 Development builds

If you are actively working with the DFHack team on testing a feature, you may want to download and install a development build. They are available via the `testing` release channel on Steam or can be downloaded from the build artifact list on GitHub for specific repository commits.

To download a development build from GitHub:

- Ensure you are logged into your GitHub account
- Go to https://github.com/DFHack/dfhack/actions/workflows/build.yml?query=branch%3Adevelop+event%3Apush
- Click on the first entry that has a green checkmark
- Click the number under "Artifacts" (or scroll down)
- Click on the "dfhack-*build*-" artifact for your platform to download

You can extract this package the same as if you are doing a manual install (see the next section).

3.2.3 Older releases

If you are downloading DFHack for very old versions of DF, the binaries for 0.40.15-r1 to 0.34.11-r4 are on DFFD. Even older versions are available here.

3.3 Installing DFHack

If you are installing from Steam, this is handled for you automatically. The instructions here are for manual installs.

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.

If you are on Windows, please remember to right click on the file after downloading, open the file properties, and select the "Unblock" checkbox. This will prevent issues with Windows antivirus programs.

The release archives contain a `hack` folder where DFHack binary and system data is stored, a `stonesense` folder that contains data specific to the *stonesense* 3d renderer, and various libraries and executable files. 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 `save` folder, among other things. Some redistributions of Dwarf Fortress may place DF in another folder, so ensure that the `hack` folder ends up next to the `data` folder, and you'll be fine.

3.3.1 Installing into a wineskin on Mac

Until DF (and DFHack) is natively available for Mac, you'll have to run the Windows version under emulation. Here are the instructions for adding DFHack to a wineskin that has DF installed in it:

1. Find the location of your existing Dwarf Fortress app (default is `/user/applications/Wineskin/`). Control + click and select "Show package contents" from the menu.
2. Find the location of the `Dwarf Fortress` folder inside the package contents (default is `/drive_c/Program Files/`)
3. Copy the contents of the unzipped DFHack folder (Windows versino) into the `Dwarf Fortress` folder inside the package.

These instructions were last tested on Mac Sonoma 14.1.2.

3.4 Uninstalling DFHack

Just renaming or removing the `dfhooks` library file is enough to disable DFHack. If you would like to remove all DFHack files, consult the DFHack install archive to see the list of files and remove the corresponding files in the Dwarf Fortress folder. Any DFHack files left behind will not negatively affect DF.

On Steam, uninstalling DFHack will cleanly remove everything that was installed with DFHack, so there is nothing else for you to do.

Note that Steam will leave behind the `dfhack-config` folder, which contains all your personal DFHack-related settings and data. If you keep this folder, all your settings will be restored when you reinstall DFHack later.

3.5 Upgrading DFHack

Again, if you have installed from Steam, your copy of DFHack will automatically be kept up to date. This section is for manual installers.

First, remove the `hack` and `stonesense` folders in their entirety. This ensures that files that don't exist in the latest version are properly removed and don't affect your new installation.

Then, follow the instructions in the *Installing DFHack* section above, making sure to choose to overwrite any remaining top-level files when extracting.

CHAPTER

FOUR

DFHACK CORE

Contents

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

All tools distributed with DFHack are documented *here*.

4.2 Using DFHack commands

DFHack commands can be executed in a number of ways:

1. Typing the command into the DFHack console (see below)
2. From the OS terminal (see below)
3. Pressing a key combination set up with *keybinding*
4. From one of several *Init files*, automatically
5. Using *script* to run a batch of commands from a file
6. From an in-game command launcher interface like *gui/launcher*, the *hotkeys* overlay widget, or *gui/quickcmd*.

4.2.1 The DFHack console

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

To include whitespace in the argument/s to some command, quote it in double quotes. To include a double quote character, use \".

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

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

4.2.2 Using an OS terminal

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

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

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

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

dfhack-run

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

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

Examples:

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

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

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

4.3 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.

4.3.1 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 two places each time they could be run:

1. The `dfhack-config/init` subdirectory in the main DF directory and

2. `save/world/init`, where `{world}` is the current save

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 `save` 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 tools 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.

init.d/*.lua

Any lua script named `init.d/*.lua`, in the save or main DF directory, will be run when any world or that save is loaded.

4.3.2 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. `dfhack-config/scripts`
2. `save/world/scripts` (only if a save is loaded)
3. `hack/scripts`
4. `data/installed_mods/...` (see below)

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

Scripts in installed mods

Scripts in mods are automatically added to the script path. The following directories are searched for mods:

```
../../workshop/content/975370/ (the DF Steam workshop directory)
mods/
data/installed_mods/
```

Each mod can have two directories that contain scripts:

- **`scripts_modactive/` is added to the script path if and only if the mod is** active in the loaded world.
- **`scripts_modinstalled/` is added to the script path as long as the mod is** installed in one of the searched mod directories.

Multiple versions of a mod may be installed at the same time. If a mod is active in a loaded world, then the scripts for the version of the mod that is active will be added to the script path. Otherwise, the latest version of each mod is added to the script path.

Scripts for active mods take precedence according to their load order when you generated the current world.

Scripts for non-active mods are ordered by their containing mod's ID.

For example, the search paths for mods might look like this:

```
activemod_last_in_load_order/scripts_modactive
activemod_last_in_load_order/scripts_modinstalled
activemod_second_to_last_in_load_order/scripts_modactive
activemod_second_to_last_in_load_order/scripts_modinstalled
...
inactivemod1/scripts_modinstalled
inactivemod2/scripts_modinstalled
...
```

Not all mods will have script directories, of course, and those mods will not be added to the script search path. Mods are re-scanned whenever a world is loaded or unloaded. For more information on scripts and mods, check out the *DFHack modding guide*.

Custom script paths

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

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

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

Tip

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

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

4.4 Commandline options

In addition to *Using an OS terminal* to execute commands on startup, DFHack also recognizes a few commandline options.

4.4.1 Options passed to Dwarf Fortress

These options can be passed to Dwarf Fortress and will be intercepted by DFHack. If you are launching Dwarf Fortress from Steam, you can set your options in the "Launch Options" text box in the properties for the Dwarf Fortress app (**NOT the DFHack app**). Note that these launch options will be used regardless of whether you run Dwarf Fortress from its own app or DFHack's.

- `--disable-dfhack`: If set, then DFHack will be disabled for the session. You will have to restart Dwarf Fortress without specifying this option in order to use DFHack. Note that even if DFHack is disabled, `stdout.txt` and `stderr.txt` will still be redirected to `stdout.log` and `stderr.log`, respectively.
- `--nosteam-dfhack`: If set, then the DFHack stub launcher will not execute when you launch DF from its own app in the Steam client. This will prevent your settings from being restored or backed up with Steam Cloud Save. This is probably not what you want. If you want to just not have the DFHack playtime counted towards your hours, see the DFHack stub launcher `--nowait` option below.

4.4.2 Options passed to the DFHack Steam stub launcher

These options can be passed to the DFHack stub launcher that executes when you run the DFHack app from the Steam client. You can set your options in the "Launch Options" text box in the properties for the DFHack app (**NOT the Dwarf Fortress app**). Note that these launch options will be used regardless of whether you run Dwarf Fortress from its own app or DFHack's.

- `--nowait`: If set, the DFHack stub launcher will not wait for DF to exit before exiting itself. This may be desired by players who do not want their playtime "double counted". However, using this option means that your DFHack settings that get backed up to the cloud will always be out of sync. The stub launcher normally downloads updated

settings from Steam Cloud Save when DF launches, and then backs up changed settings when DF exits. If this option is used, then your settings will still be reconciled when DF launches, but changes made during your play session will not be saved when DF exits. Please use with caution – you may lose data.

4.5 Environment variables

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

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

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

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

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

4.6 Core preferences

There are a few settings that can be changed dynamically via *gui/control-panel* to affect runtime behavior. You can also toggle these from the commandline using the *lua* command, e.g. `lua dfhack.HIDE_ARMOK_TOOLS=true` or by editing the generated `dfhack-config/init/dfhack.control-panel-preferences.init` file and restarting DF.

- `dfhack.HIDE_CONSOLE_ON_STARTUP`: Whether to hide the external DFHack terminal window on startup. This, of course, is not useful to change dynamically. You'll have to use *gui/control-panel* or edit the init file directly and restart DF for it to have an effect.
- `dfhack.HIDE_ARMOK_TOOLS`: Whether to hide "armok" tools in command lists.

4.7 Performance monitoring

Though DFHack tools are generally performant, they do take some amount of time to run. DFHack tracks its impact on DF game speed so the DFHack team can be aware of (and fix) tools that are taking more than their fair share of processing time.

The target threshold for DFHack CPU utilization during unpaused gameplay with all overlays and automation tools enabled is 10%. This is about the level where players would notice the impact. In general, DFHack will have even less impact for most players, since it is not common for every single DFHack tool to be enabled at once.

DFHack will record a performance report with the savegame files named `dfhack-perf-counters.dat`. The report contains measurements from when the game was loaded to the time when it was saved. By default, only unpaused time is measured (since processing done while the game is paused doesn't slow anything down from the player's perspective). You can display a live report at any time by running:

```
:lua require('script-manager').print_timers()
```

You can reset the timers to start a new measurement session by running:

```
:lua dfhack.internal.resetPerfCounters()
```

If you want to record performance over all elapsed time, not just unpaused time, then instead run:

```
:lua dfhack.internal.resetPerfCounters(true)
```

CHAPTER

FIVE

DFHACK TOOLS

DFHack comes with **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

5.1 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.

5.2 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.

Some tools are part of our back catalog and haven't been updated yet for v50 of Dwarf Fortress. These tools are tagged as unavailable. They will still appear in the alphabetical list at the bottom of this page, but unavailable tools will not listed in any of the indices.

5.3 DFHack tools by game mode

- **adventure**
  Tools that are useful while in adventure mode.
- **dfhack**
  Tools that you use to run DFHack commands or interact with the DFHack or DF system.
- **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.

5.4 DFHack tools by theme

- **armok**
  Tools which give the player god-like powers or the ability to access information the game intentionally keeps hidden. Players that do not wish to see these tools can hide them in the `Preferences` tab of *gui/control-panel*.
- **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 with fort layout.
- **dev**
  Tools that are useful when debugging or developing mods.
- **fps**
  Tools that help you prevent impact to your FPS.
- **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 perform common tasks quickly and easily.

5.5 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.

5.6 All DFHack tools alphabetically

5.6.1 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.

5.6.2 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.

5.6.3 adaptation

Tags: fort | armok | units

Command: `adaptation`

Adjust a unit's cave adaptation level.

Cave adaptation (or adaption) increases for a unit when they spend time underground. If it reaches a high enough level, the unit will be affected when View or set the level of cavern adaptation for the selected unit or the whole fort.

Usage

```
adaptation [show] [--all]
adaptation set [--all] <value>
```

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

Examples

`adaptation`
Show the cave adaptation level for the selected unit.

`adaptation set --all 0`
Clear the cave adaptation levels for all citizens and residents.

Options

`-a`, `--all`
Apply to all citizens and residents.

5.6.4 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, for example, not all civs get high boots. This script can help fix that. Only weapons, armor, and tools are currently supported; dynamically generated item types like instruments are not.

Usage

```
add-recipe (all|native)
add-recipe single <item token>
```

Examples

`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 SHOES:ITEM_SHOES_BOOTS`
: Allow your civ to craft high boots.

5.6.5 add-spatter

Tags: adventure | fort | gameplay | items

Add poisons and magical effects to weapons.

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.

5.6.6 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.

5.6.7 adv-fix-sleepers

Tags: unavailable

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
```

5.6.8 adv-max-skills

Tags: unavailable

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
```

5.6.9 adv-rumors

Tags: unavailable

Command: `adv-rumors`

Improves the rumors menu in adventure mode.

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
```

5.6.10 agitation-rebalance

Tags: fort | gameplay

Command: `agitation-rebalance`

Make agitated wildlife and cavern invasions less persistent.

The DF agitation (or "irritation") system gives you challenges to face when your dwarves impact the natural environment. It adds new depth to gameplay, but it can also quickly drag the game down, both with constant incursions of agitated animals and with FPS-killing massive buildups of hidden invaders in the caverns. This mod changes how the agitation system behaves to ensure the challenge remains fun and scales appropriately according to your dwarves' current activities.

In short, this mod changes irritation-based attacks from a constant flood to a system that is responsive to your recent actions on the surface and in the caverns. If you irritate the natural environment enough, you will be attacked exactly once. You will not be attacked further unless you continue to antagonize nature. This mod can be enabled (and auto-started for new forts, if desired) on the "Gameplay" tab of *gui/control-panel*.

Usage

```
enable agitation-rebalance
agitation-rebalance [status]
agitation-rebalance preset <name>
agitation-rebalance enable|disable <feature>
```

When run without arguments (or with the `status` argument), it will print out whether it is enabled, the current configuration, how many agitated creatures and (visible) cavern invaders are on the map, and your current chances of suffering retaliation on the surface and in each of the cavern layers.

The *Presets* allow you to quickly set the game irritation-related difficulty settings to tested, balanced values. You can adjust them further (or set your own values) on the DF difficulty settings screen. Note that `agitation-rebalance preset` can be used to set the difficulty settings even if the mod is not enabled. Even with vanilla mechanics, the presets are still handy.

Finally, each feature of the mod can be individually enabled or disabled. More details in the *Features* section below.

Examples

`agitation-rebalance preset lenient`
: Load the `lenient` preset, which allows for a fair amount of tree cutting and other activity between attacks. This preset is loaded automatically if the `auto-preset` feature enabled (it's enabled by default) and you have the "Enemies" difficulty settings at their default "Normal" values.

`enable agitation-rebalance`
: Manually enable the mod (not needed if you are using *gui/control-panel*)

`agitation-rebalance enable monitor`
: Enables an overlay that shows the current chances of being attacked on the surface or in the caverns. The danger ratings shown on the overlay are accurate regardless of whether `agitation-rebalance` is enabled, so you can use the monitor even if you're not using the mod.

How the DF agitation system works

The surface

For the surface wilderness in savage biomes (non-savage biomes will never see agitated wildlife), DF maintains a counter. Your dwarves increase the counter when they chop down trees or catch fish. Once it crosses a threshold, wildlife that enters the map has a chance of becoming agitated and aggressively attacking your units. This chance increases the higher the counter rises. Once a year, the counter is decremented by a fixed amount.

Only one group of wildlife can be on the surface at a time. When you kill all the creatures in a group, or when they finally wander off on their own, the game spawns a new group to replace them. Each new group rolls against the current surface irritation counter for a chance to become agitated.

Since agitated wildlife seeks out your units to attack, they are often quickly destroyed – if they don't quickly destroy *you* – and a new wave will spawn. The new wave will have a similar chance to the previous wave for becoming agitated. This means that once you cross the threshold that starts the agitation attacks, you may suffer near-constant retribution until you stop all tree cutting and fishing on the surface and hide for sufficient time (years) until the counter falls low enough again.

The caverns

DF similarly maintains counters for each cavern layer, with chances of cavern invasion and forgotten beast attack independently checked once per season. The cavern irritation counter is increased for tree felling and fishing within the cavern. Moreover, doing anything that makes noise will increase the irritation. For example, digging anywhere within the cavern's z-level range (even if it is not in the open space of the cavern itself) will raise the cavern's irritation level.

The chance of cavern invasion increases linearly with respect to irritation until it reaches 100% after about 100 cavern trees felled. While irritation chances are calculated separately for each cavern layer, only one attack may occur per season. The upper cavern layers get rolled first, so even if all layers have the same irritation level, invasions will tend to happen in the uppermost layer. There are no player-configurable settings to change the cavern invasion thresholds. Regardless of irritation level, cavern invasions do not spawn until the cavern layer is discovered by the current fort.

The chance of forgotten beast attack in a particular layer is affected by the cavern layer's irritation level, but your fortress's wealth has a much greater impact. Even with an irritation level of zero, a wealthy fortress will encourage forgotten beasts to attack at their maximum rate. The chance of forgotten beast attack is capped at 33% per layer, but unlike cavern invasions, you can have as many forgotten beast attacks in a season as you have layers. With high irritation and/or high fortress wealth, forgotten beasts can invade a cavern before you discover it.

You can wall off the caverns to insulate your fort from the invasions, but invaders will continue to spawn and build up over time. Cavern invaders spawn hidden, so you will not be aware that they are there until you send a unit in to investigate. Eventually, your FPS will be impacted by the large numbers of cavern invaders. The irritation counters for the cavern layers do not decay over time, so once attacks begin, cavern invasions will occur once a season thereafter, regardless of the continued presence of previous invaders.

Irritation counters are saved with the cavern layer in the world region, which extends beyond the boundaries of your current fort. If you retire a fort and start another one nearby, the caverns will retain any irritation added by the first fort. This means that new forts may start with already-irritated caverns and meet with immediate resistence.

The settings

There are several variables that affect the behavior of this system, all customizable in the DF difficulty settings:

`Wilderness irritation minimum`
: While the surface irritation counter is below this value, no agitated wildlife will appear.

`Wilderness sensitivity`
: After the surface irritation counter rises above the minimum, this value represents the range over which the chance of attack increases from 0% to 100%.

`Wilderness irritation decay`
: This is the amount that the surface irritation counter decreases per year, regardless of activity. Due to a bug in DF, the widget for this setting in the difficulty settings panel always displays and controls the value for `Wilderness irritation minimum` and thus the setting cannot be changed in the vanilla interface from its default value of 500 (if initialized by the "Normal" vanilla preset) or 100 (if initialized by the "Hard" vanilla preset).

`Cavern dweller maximum attackers`
: This controls the maximum number of cavern invaders that can spawn in a single invasion. If `agitation-rebalance` is not managing the invader population, the number of invaders in the caverns can grow beyond this number if the invaders from a previous invasion are still alive.

`Cavern dweller scale`
: Each time your civilization is attacked, the number of attackers in a single cavern invasion increases by this value. The total number of attackers is still capped by `Cavern dweller maximum attackers`.

`Forgotten beast wealth divisor`
: Your fortress wealth is divided by this number and the result is added to a cavern's "natural" irritation to get the effective irritation that a forgotten beast rolls against for a chance to attack.

`Forgotten beast irritation minimum`
: While a cavern's effective irritation (see `Forgotten beast wealth divisor`) is below this value, no forgotten beasts will invade that cavern.

`Forgotten beast sensitivity`
: After the cavern's effective irritation rises above the minimum, this value represents the range over which the chance of forgotten beast attack increases from 0% to 100%.

What does this mod do?

When enabled, this mod makes the following changes:

When agitated wildlife enters the map on the surface, the surface irritation counter is set to the value of `Wilderness irritation minimum`, ensuring that the *next* group of widlife that enters the map will *not* be agitated. This means that the incursions act more like a warning shot than an open floodgate. You will not be attacked again unless you continue your activities on the surface that raise the chance of a subsequent attack.

The larger the value of `Wilderness sensitivity`, the more you can irritate the surface before you suffer another incursion. For reference, each tree chopped adds 100 to the counter, so a `Wilderness irritation minimum` value of 3500 and a `Wilderness sensitivity` value of 10000 will allow you to initially chop 35 trees before having any chance of being attacked by agitated creatures. Each tree you chop beyond those initial 35 raises the chance that the next wave of wildlife will be agitated by 1%.

If you cross a year boundary, then you will have additional leniency granted by the `Wilderness irritation decay` value (if it is set to a value greater than zero).

For the caverns, we don't want to adjust the irritation counters directly since that would negatively affect the chances of being attacked by (the much more interesting) forgotten beasts. Instead, when a cavern invasion begins, we record the current irritation counter value and effectively use that as the new "minimum". A "sensitivity" value is synthesized

from the average of the values of `Wilderness irritation minimum` and `Wilderness sensitivity`. This makes cavern invasions behave similarly to surface agitation and lets it be controlled by the same difficulty settings. The parameters for forgotten beast attacks can still be controlled independently of this mod.

Finally, if you have walled yourself off from the danger in the caverns, yet you continue to irritate nature down there, this mod will ensure that the number of active cavern invaders, cumulative across all cavern levels, never exeeds the value set for `Cavern dweller maximum attackers`. This prevents excessive FPS loss during gameplay and keeps the number of creatures milling around outside your gates (or hidden in the shadows) to a reasonable number.

The monitor

You can optionally enable a small monitor panel that displays the current threat rating for an upcoming attack. The chance of being attacked is shown for the surface and for the caverns as a whole (so as not to spoil exactly where the attack will happen). Moreover, to avoid spoiling when a cavern invasion has begun, the displayed threat rating for the caverns is not reset to "None" (or, more likely, "Low", since the act of fighting the invaders will have raised the cavern's irritation a bit) until you have discovered and neutralized the invaders.

The ratings shown on the overlay are accurate regardless of whether `agitation-rebalance` is enabled. That is, if this mod is not enabled, then the monitor will display ratings according to vanilla mechanics.

Presets

The tree counts in these presets are only estimates. There are other actions that contribute to irritation other than chopping trees, like fishing. Noise also contributes to irritation in the caverns. However, tree chopping is the most important factor.

`casual`

- Trees until chance of invasion: 1000
- Surface invasion chance increase per additional tree: 0.1%
- Additional allowed trees per year: 1000
- Trees until risk of next cavern invasion: 1000
- Max cavern invaders: 0

`lenient`

- Trees until chance of invasion: 100
- Surface invasion chance increase per additional tree: 1%
- Additional allowed trees per year: 50
- Trees until risk of next cavern invasion: 100
- Max cavern invaders: 20

`strict`

- Trees until chance of invasion: 25
- Surface invasion chance increase per additional tree: 20%
- Additional allowed trees per year: 10
- Trees until risk of next cavern invasion: 15
- Max cavern invaders: 50

`insane`

- Trees until chance of invasion: 6
- Surface invasion chance increase per additional tree: 50%
- Additional allowed trees per year: 2
- Trees until risk of next cavern invasion: 4
- Max cavern invaders: 100

After using any of these presets, you can always to go the vanilla difficulty settings and adjust them further to your liking.

If the `auto-preset` feature is enabled and the difficulty settings exactly match any of the vanilla "Enemies" presets when the mod is enabled, a corresponding mod preset will be loaded. See the *Features* section below for details.

Features

Features of the mod can be individually enabled or disabled. All features except for `monitor` are enabled by default. Available features are:

`auto-preset`
: Auto-load a preset based on which vanilla "Enemies" preset is active: - "Off" loads the "casual" preset - "Normal" loads the "lenient" preset - "Hard" loads the "strict" preset This feature takes effect at the time when the mod is enabled, so if you don't want your default vanilla settings changed, be sure to disable this feature before enabling `agitation-rebalance`.

`surface`
: Manage surface agitated wildlife frequency.

`cavern`
: Manage cavern invasion frequency.

`cap-invaders`
: Ensure the number of live invaders in the caverns does not exceed the configured maximum.

`monitor`
: Display a panel on the main map showing your chances of an irritation-related attack on the surface and in the caverns. See *The monitor* section above for details. The monitor overlay can also be enabled and disabled via *gui/control-panel*, or repositioned with *gui/overlay*.

Caveat

If a cavern invasion causes the number of active attackers to exceed the maximum, this mod will gently redirect the excess cavern invaders towards oblivion as they enter the map. You may notice some billowing smoke near the edge of the map as the surplus invaders are lovingly vaporized.

5.6.11 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
```

5.6.12 allneeds

Tags: fort | units

Command: `allneeds`

Summarize the cumulative needs of a unit or the entire fort.

Provides an overview of the needs of the selected unit, or, if no unit is selected, the fort in general. By default, the list is sorted by which needs are making your dwarves (or the selected dwarf) most unfocused right now.

Usage

```
allneeds [<options>]
```

Examples

`allneeds`
Show the cumulative needs for the entire fort, or just for one unit if a unit is selected in the UI.

`allneeds --sort strength`
Sort the list of needs by how strongly the people feel about them.

Options

`-s, --sort <criteria>`
Choose the sort order of the list. the criteria can be:

- `id`: sort the needs in alphabetical order.
- `strength`: sort by how strongly units feel about the need. that is, if left unmet, how quickly the focus will decline.
- `focus`: sort by how unfocused the unmet needs are making your dwarves feel right now.
- `freq`: sort by how many times the need is seen (note that a single dwarf can feel a need many times, e.g. when needing to pray to multiple gods).

5.6.13 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

5.6.14 aquifer

Tags: fort | armok | map

Command: `aquifer`

Add, remove, or modify aquifers.

This tool can examine or alter aquifer properties of map tiles. Also see *gui/aquifer* for visually highlighting active aquifer tiles and interactively modifying them.

Only solid (wall), natural (not constructed), rough (not smooth) layer stone (not mineral vein or gem cluster) tiles can have their aquifer properties modified. When adding aquifers, walls that would immediately leak into open space will not be modified unless the override option is given.

Usage

```
aquifer [list] [<pos> [<pos>] | <target option>] [<options>]
aquifer add (light|heavy) (<pos> [<pos>] | <target option>) [<options>]
aquifer drain [light|heavy] (<pos> [<pos>] | <target option>) [<options>]
aquifer convert (light|heavy) (<pos> [<pos>] | <target option>) [<options>]
```

For each action, you can either specify a coordinate range or a target option. If specifying a coordinate range, each <pos> parameter can be either an xyz coordinate triple (e.g. `14,25,143`) or the keyword `here`, which indicates the position of the keyboard cursor. If not specified, the second position of the coordinate range defaults to `here` if you have the keyboard cursor active or is equal to the first coordinate otherwise.

Examples

aquifer
List z-levels (and their associated displayed elevations) that have aquifer tiles and how many aquifer tiles are on each listed level.

aquifer drain --all --skip-top 2
Drain all aquifer tiles on the map except for the top 2 levels of aquifer. This example is available as a Gameplay -> Autostart option in *gui/control-panel*.

aquifer drain -z --leaky
Drain aquifer tiles on this z-level that are actively leaking into open adjacent tiles or into empty spaces underneath them.

aquifer convert light --zdown --levels 5
Convert heavy aquifers from the current z-level to 4 levels below to light aquifers.

aquifer add heavy here --leaky
Create a heavy aquifer tile in the exposed natural rough wall under the keyboard cursor.

aquifer add light 87,29,126 111,53,126
Create a 25x25 block of light aquifer tiles at the specified coordinates.

aquifer add light --all --skip-top 2 --levels 20
Add a 20-level deep light aquifer starting 2 levels beneath the surface.

Target options

-a, --all
Apply the action to every possible tile in the entire map.

-d, --zdown
Apply the action to the current z-level and below.

-u, --zup
Apply the action to the current z-level and above.

-z, --cur-zlevel
Apply the action to the current z-level only.

Other options

-l, --levels <num>
If one of the `--all`, `--zdown`, or `--zup` target options are specified, this option will limit the number of levels that will be affected. If used with `--all`, the levels are counted from the topmost level that can hold an aquifer tile.

--leaky
Aquifer tiles will leak into orthogonally adjacent open spaces or into empty space directly beneath them. These kinds of tiles are not modified by default by the `add` action. This option will bypass that safety check and allow addition of aquifer tiles that will immediately begin leaking. If used with the `drain` or `convert` actions, the action will only be applied to tiles that are actively leaking.

-q, --quiet
Don't print any non-error output.

-s, --skip-top <num>
If the `--all` target option is specified, this option will ignore the first <num> levels before the specified action

starts taking effect. For the `add` action, it keeps the first `<num>` levels starting from the topmost surface elevation unchanged before adding aquifers. For the `drain` and `convert` actions, it keeps the top `<num>` levels starting from the topmost existing aquifer unchanged before removing or converting aquifers.

5.6.15 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.

5.6.16 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

5.6.17 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.

5.6.18 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.

5.6.19 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.

5.6.20 assign-minecarts

Tags: fort | productivity

Command: `assign-minecarts`

Assign minecarts to hauling routes.

This script allows you to quickly assign minecarts to hauling routes without having to go through 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
```

5.6.21 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, olives and yak":

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

- "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.

5.6.22 assign-profile

Tags: unavailable

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.

5.6.23 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

5.6.24 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. Units will be ignored if they are:

- Untamed
- Named or nicknamed
- Caged, if and only if the cage is in a zone (to protect zoos)
- Trained for war or hunting

Creatures that are least useful for animal breeding programs are marked for slaughter first. That is:

- 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
- Creatures that are only partially trained will be butchered before those who are fully domesticated.
- Older adults will be butchered before younger adults.
- Younger juveniles will be butchered before juveniles that are closer to becoming adults.

The default targets are: 2 male kids, 4 female kids, 2 male adults, and 4 female adults. Note that you may need to set a target above 1 to have a reliable breeding population due to asexuality etc.

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

`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. This option is enabled by default.

`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 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 now
Process all livestock according to the current watchlist configuration, even if the plugin is not currently enabled, and thus not doing automatic periodic scans.

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 10 10 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
```

5.6.25 autochop

Tags: fort | auto | plants

Command: `autochop`

Auto-harvest trees when low on stockpiled logs.

This plugin can automatically designate trees for chopping when your stocks are low on logs. It can also ensure specified burrows are kept clear of trees, so, for example, caravans always have a path to your depot and trees won't grow close to your walls, giving invaders an unexpected path into your fort.

Autochop checks your stock of logs and designates appropriate trees for chopping once every in-game day. Logs that are forbidden or inaccessible (e.g. in hidden parts of the map, under water, etc.) are not counted towards your target. Trees that are inaccessible are likewise never designated.

Please see *gui/autochop* for the interactive configuration dialog.

Usage

```
enable autochop
autochop [status]
autochop (designate|undesignate)
autochop target <max> [<min>]
autochop (chop|nochop) <burrow>[,<burrow>...]
autochop (clearcut|noclearcut) <burrow>[,<burrow>...]
autochop (protect|unprotect) <type>[,<type>...] <burrow>[,<burrow>...]
```

Examples

Ensure we always have about 200 logs in stock, harvested from accessible trees anywhere on the map:

```
enable autochop
```

Ensure we always have about 500 logs in stock, harvested from a burrow named "TreeFarm". Also ensure the caravan pathway and the area around the outer wall ("CaravanPath" and "OuterWall") are always clear:

```
enable autochop
autochop target 500
autochop chop TreeFarm
autochop clearcut CaravanPath,OuterWall
```

Clear all non-food-producing trees out of a burrow ("PicnicArea") intended to contain only food-producing trees:

```
autochop clearcut PicnicArea
autochop protect brewable,edible,cookable PicnicArea
autochop designate
```

Commands

status
Show current configuration and relevant statistics.

designate
Designate trees for chopping right now according to the current configuration. This works even if autochop is not currently enabled.

undesignate
Undesignates all trees.

target <max> [<min>]
Set the target range for the number of logs you want to have in stock. If a minimum amount is not specified, it defaults to 20% less than the maximum. The default target is 200 (with a corresponding minimum of 160).

(no)chop <burrow>[,<burrow>...]
Instead of choosing trees across the whole game map, restrict tree cutting to the given burrows. Burrows can be specified by name or internal ID.

(no)clearcut <burrow>[,<burrow>...]
Ensure the given burrows are always clear of trees. As soon as a tree appears in any of these burrows, it is designated for chopping at priority 2.

(un)protect <type>[,<type>...] <burrow>[,<burrow>...]
Choose whether to exclude trees from chopping that produce any of the given types of food. Valid types are: brewable, edible, and cookable.

5.6.26 autoclothing

Tags: fort | auto | workorders

Command: autoclothing

Automatically manage clothing work orders.

This command allows you to configure a "uniform" that your civilians should wear. autoclothing will then keep track of your stock of the configured clothing (including clothing that your citizens are already wearing) and will generate manager orders to manufacture more when your stock drops below a configured per-citizen threshold.

If you are wondering whether you should enable *tailor* instead, see the comparsion in the last section below.

Usage

```
autoclothing
autoclothing <material> <item>
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".

The quantity is **per-citizen**, so keep the values low. A quantity of 1 makes one per citizen. 2 will ensure everyone has a pre-made spare. Usually, 1 is enough.

When invoked without parameters, it shows a summary of all managed clothing orders, and the overall clothing situation in your fort. 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" 1`
: Ensures that every citizen will have a cloth short skirts available (as long as there is cloth available to make them out of).

`autoclothing cloth dress`
: Displays the currently set number of cloth dresses chosen per citizen.

Which should I enable: autoclothing or tailor?

Both `autoclothing` and *tailor* generate manager orders for needed clothing, but they make different choices about when and what to order.

Enable `autoclothing` when:

- you want to set specific configuration for each clothing type for more control over exactly what your citizens wear
- you want to keep a cache of spare clothing before it is needed

Enable *tailor* when:

- you want your citizens clothed and don't care specifically what they wear
- you want a tool that can run effectively with a default configuration

You can even enable both tools if you only want to set configuration for a few specific clothing types (e.g. if you'd prefer that most pople wear dresses). You can set the configuration for those types in `autoclothing` and let *tailor* automatically manage the rest.

5.6.27 autodump

Tags: fort | armok | fps | items

Command: `autodump`

Instantly gather or destroy items marked for dumping.

Command: `autodump-destroy-here`

Destroy items marked for dumping under the keyboard cursor.

This tool can instantly move all unforbidden items marked for dumping to the tile under the keyboard 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 keyboard cursor must be placed on a floor tile so the items can be dumped there.

Usage

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

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

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-here`
Destroys items on the selected tile that are marked for dumping.

5.6.28 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

5.6.29 autofish

Tags: fort | auto | labors

Command: `autofish`

Auto-manage fishing labors to control your stock of fish.

This script makes managing how much fish you keep around automatic. It tries to maintain a configured stock level of raw and/or prepared fish, partially to keep item quantities from ballooning out of control, and partly to try and prevent collecting too many rotten fish.

Usage

enable autofish
Enable the script

disable autofish
Disable the script

autofish status
Show the current status of the script, your configured values, and whether or not fishing is currently enabled.

autofish <max> [min] [<options>]
Change autofish settings.

Positional Parameters

max
(default: 100) controls the maximum amount of fish you want to keep on hand in your fortress. Fishing will be disabled when the amount of fish goes above this value.

min
(default: 75) controls the minimum fish you want before restarting fishing.

Options

r, --raw (true | false)
(default: `true`) Set whether or not raw fish should be counted in the running total of fish in your fortress.

Examples

enable autofish
Enables the script.

autofish 150 -r true
Sets your maximum fish to 150, and enables counting raw fish.

autofish 300 250
Sets your maximum fish to 300 and minimum to 250.

5.6.30 autogems

Tags: unavailable

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`).

5.6.31 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: This plugin is still being tested. Use at your own risk.

The algorithms that autolabor uses to choose labor assignments have *not* been updated for version 50 of Dwarf Fortress. There is no particular guarantee that the labor assignments autolabor is making are optimal, and it is entirely possible that the assignments it makes will lead to unforeseen consequences. You should monitor what your dwarves are doing, and more importantly not doing, when using autolabor.

At this time there is no way to easily see what labors are being assigned to whom, as there is no longer any vanilla means for seeing the labor assignment table. Until *manipulator* is once again available, probably the best way to see what autolabor is doing is to use Dwarf Therapist. You can also increase autolabor's logging level using the *debugfilter* command (setting either `debug` or `trace` level for the `cycle` mode) but be warned that this may generate a large amount of console spam, especially in a large fort.

When it is enabled, autolabor automatically disables the work detail system. You cannot use autolabor and work details at the same time. If you attempt to open the work detail screen while autolabor is active, a warning box should appear advising you that autolabor is managing labors and preventing you from making any changes on that screen.

Finally, should you disable autolabor, autolabor will automatically reenable the vanilla work detail system. However, the work detail system only updates labors when the work detail screen is open and some change is made on that screen. Therefore, if you choose to disable autolabor, you should probably immediately thereafter open the work details screen and make some change to force the game to recompute all labor assignments based on the vanilla algorithm. At this time, it is not possible for autolabor to do this automatically.

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.

5.6.32 autolabor-artisans

Tags: unavailable

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.

5.6.33 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.

5.6.34 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!

5.6.35 autoslab

Tags: fort | auto | workorders

Automatically engrave slabs for ghostly citizens.

Automatically queue orders to engrave slabs of existing ghosts. Will only queue an order if there is no existing slab with that unit's memorial engraved and there is not already an existing work order to engrave a slab for that unit. Make sure you have spare slabs on hand for engraving! If you run *orders import library/rockstock*, you'll be sure to always have some slabs in stock.

Usage

`enable autoslab`
: Enables the plugin and starts checking for ghosts that need memorializing.

`disable autoslab`
: Disables the plugin.

5.6.36 ban-cooking

Tags: fort | productivity | items | plants

Command: `ban-cooking`

Protect useful items from being cooked.

Some cookable ingredients have other important uses. For example, seeds can be cooked, but if you cook them all, then your farmers will have nothing to plant in the fields. Similarly, booze can be cooked, but if you do that, then your dwarves will have nothing (good) to drink.

If you open the Kitchen screen, you can select individual item types and choose to ban them from cooking. To prevent all your booze from being cooked, for example, you'd filter by "Drinks" and then click each of the visible types of booze to prevent them from being cooked. Only types that you have in stock are shown, so if you acquire a different type of booze in the future, you have to come back to this screen and ban the new types.

Instead of doing all that clicking, `ban-cooking` can ban entire classes of items (e.g. all types of booze) in one go. It can even ban types that you don't have in stock yet, so when you *do* get some in stock, they will already be banned. It will never ban items that are only good for eating or cooking, like meat or non-plantable nuts. It is usually a good idea to run `ban-cooking all` as one of your first actions in a new fort. You can add this command to your Autostart list in *gui/control-panel*.

If you want to re-enable cooking for a banned item type, you can go to the Kitchen screen and un-ban whatever you like by clicking on the "cook" icon. You can also un-ban an entire class of items with the `--unban` option.

Usage

```
ban-cooking <type|all> [<type> ...] [<options>]
```

Valid types are:

- `booze`
- `brew` (brewable plants)
- `fruit`
- `honey`
- `milk`
- `mill` (millable plants)
- `oil`
- `seeds` (plantable seeds)
- `tallow`
- `thread`

Note that in the vanilla game, there are no items that can be milled or turned into thread that can also be cooked, so these types are only useful when using mods that add such items to the game.

Examples

`ban-cooking oil tallow`
Ban all types of oil and tallow from cooking.

`ban-cooking all`
Ban all otherwise useful types of foods from being cooked. This command can be enabled for Autostart in *gui/control-panel*.

Options

`-u`, `--unban`
Un-ban the indicated item types.

`-v`, `--verbose`
Print each ban as it happens.

5.6.37 binpatch

Tags: unavailable

Command: `binpatch`

Applies or removes binary patches.

See *Patching the DF binary* for more info.

Usage

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

5.6.38 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 `dfhack-config/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.

Note

blueprint is still in the process of being updated for the new version of DF. Stockpiles (the "place" phase), zones (the "zone" phase), building configuration (the "query" phase), and game configuration (the "config" phase) are not yet supported.

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.

5.6.39 bodyswap

Tags: unavailable

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.

5.6.40 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

5.6.41 break-dance

Tags: unavailable

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
```

5.6.42 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 keyboard 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).

`-z`, `--zlevel`
Restrict operation to the currently visible z-level

5.6.43 building-hacks

Tags: unavailable

Provides a Lua API for creating powered workshops.

See *building-hacks* for more details.

5.6.44 buildingplan

Tags: fort | design | productivity | buildings

Command: `buildingplan`

Plan building layouts with or without materials.

Buildingplan allows you to place furniture, constructions, and other buildings, regardless of whether the required materials are available. This allows you to focus purely on design elements when you are laying out your fort, and defers item production concerns to a more convenient time.

Buildingplan is an alternative to the vanilla building placement UI. It appears after you have selected the type of building, furniture, or construction that you want to place in the vanilla build menu. Buildingplan then takes over for the actual placement step. If the placed building requires materials that aren't available yet, it will be created in a suspended state. Buildingplan will periodically scan for appropriate items and attach them to the planned building. Once all items are attached, the construction job will be unsuspended and a dwarf will come and build the building. If you have the *unsuspend* overlay enabled (it is enabled by default), then buildingplan-suspended buildings will be tagged with a clock graphic in graphics mode or a `P` marker in ASCII mode, as opposed to the `x` marker for "regular" suspended buildings. If you have *suspendmanager* running, then buildings will be left suspended when their items are all attached and `suspendmanager` will unsuspend them for construction when it is safe to do so.

If you want to impose restrictions on which items are chosen for the buildings, buildingplan has full support for quality and material filters (see below). This lets you create layouts with a consistent color, if that is part of your design.

If you just care about the heat sensitivity of the building, you can set the building to be fire- or magma-proof in the placement UI screen. This makes it very easy to ensure that your pump stacks and floodgates, for example, are magma-safe.

Buildingplan works well in conjuction with other design tools like *gui/quickfort*, which allow you to apply a building layout from a blueprint. You can apply very large, complicated layouts, and the buildings will simply be built when your dwarves get around to producing the needed materials. If you set filters in the buildingplan UI before applying the blueprint, the filters will be applied to the blueprint buildings, just as if you had planned them from the buildingplan placement UI.

One way to integrate buildingplan into your gameplay is to create manager workorders to ensure you always have a few blocks/doors/beds/etc. available. You can then place as many of each building as you like. Items will be used to build the planned buildings as they are produced, with minimal space dedicated to stockpiles. The DFHack *orders* library can help with setting these manager workorders up for you.

If you don't want to use the `buildingplan` interface for the building you're currently trying to place, you can hit `AltM` or click on the minimize toggle in the upper right corner of the panel. If you do not wish to ever use the `buildingplan` interface, you can turn off the `buildingplan.planner` overlay in *gui/control-panel* (on the "Overlays" tab).

Usage

```
buildingplan [status]
buildingplan set <setting> (true|false)
buildingplan reset
```

Examples

buildingplan
Print a report of current settings, which kinds of buildings are planned, and what kinds of materials the buildings are waiting for.

buildingplan set boulders false
When finding items to satisfy "building materials" requirements, don't select boulders. Use blocks or logs (if enabled) instead.

buildingplan reset
Reset all settings and filters to their defaults. This command does not affect existing planned buildings.

Global settings

The buildingplan plugin has several global settings that affect what materials can be chosen when attaching items to planned buildings:

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

reconstruct (default: true)
When you plan constructions, allow building on already-built constructions. For example, if you're planning a wall, you will be able to place it on an already-built floor. This matches vanilla behavior. However, it can be annoying that floors can be planned on top of other constructed floors, especially when you're trying to fill in "holes" in a large area of constructed flooring. Turn off to treat existing constructions as "invalid" locations for new constructions. This can help when extending existing constructions and will prevent you from wasting materials by constructing twice on a tile. Note that even if this option is disabled, you can still choose to place a construction on top of an existing construction if you just select a single 1x1 tile as your planning area.

These settings 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 enable the following two commands on the `Autostart` subtab of the `Automation` tab to always configure *buildingplan* to just use blocks for buildings and constructions:

```
buildingplan set boulders false
buildingplan set logs false
```

Building placement

Once you have selected a building type to build in the vanilla build menu, the *buildingplan* placement UI appears as an *overlay* widget, covering the vanilla building placement panel.

For basic usage, you don't need to change any settings. Just click to place buildings of the selected type and right click to exit building mode. Any buildings that require materials that you don't have on hand will be suspended and built when the items are available. The closest available material will be chosen for the building job.

When building constructions, you'll get a few extra options, like whether the construction area should be hollow or what types of stairs you'd like at the top and bottom of a stairwell. Also, unlike other buildings, it is ok if some tiles selected in the construction area are not appropriate for building. For example, if you want to fill an area with flooring, you can select the entire area, and any tiles with existing buildings or walls will simply be skipped.

Some building types will have other options available as well, such as a selector for how many weapons you want in weapon traps or whether you want to only build engraved slabs.

Setting quality and material filters

If you want to set restrictions on the items chosen to complete the planned building, you can click on the "[any material]" link next to the item name or select the item with the `q` or `Q` keys and hit `f` to bring up the filter dialog.

You can select whether the item must be decorated, and you can drag the ends of the "Item quality" slider to set your desired quality range. Note that blocks, boulders, logs, and bars don't have a quality, and the quality options are disabled for those types. As you change the quality settings, the number of currently available matched items of each material is adjusted in the materials list.

You can click on specific materials to allow only items of those materials when building the current type of building. You can also allow or disallow entire categories of materials by clicking on the "Type" options on the left. Note that it is perfectly fine to choose materials that currently show zero quantity. *buildingplan* will patiently wait for items made of materials you have selected to become available.

You can save up to 10 filters to the favorites panel, which can be useful if you find yourself frequently switching between a small number of material filters (e.g. when using differently coloured stones for different parts of the fort). The panel can be accessed by pressing `Ctrlf`. Clicking on an empty slot stores the filter for the currently selected item type to the respective slot. Clicking on a slot with a filter restores that filter to the currently selected item type. Alternatively you can navigate to a slot using `x/X` and press `y` to save or restore a filter. You can free a slot by clicking the "[x]", and you can edit the label of a saved filter by shift-clicking the label.

Choosing specific items

If you want to choose specific items instead of using the filters, click on the "Choose items" selector or hit `z` before placing the building. You can choose to be prompted for every item ("Manually") or you can have it automatically select the type of item that you last chose for this building type. The list you are prompted with is sorted by most recently used materials for that building type by default, but you can change to sort by name or by available quantity by clicking on the "Sort by" selector or hitting `R`. The configuration for whether you would like to choose specific items is saved per building type and will be restored when you plan more of that building type.

You can select the maximum quantity of a specified item by clicking on the item name or selecting it with the arrow keys and hitting `Enter`. You can instead select items one at a time by Ctrl-clicking (`ShiftRight`) to increment or Ctrl-Shift-clicking (`ShiftLeft`) to decrement.

Once you are satisfied with your choices, click on the large green button or hit `C` to continue building. Note that you don't have to select all the items that the building needs. Any remaining items will be automatically chosen from other available items (or from items produced in the future if not all items are available yet). If there are multiple item types to choose for the current building, one dialog will appear per item type.

Building status

When viewing a planned building, a separate *overlay* widget appears on the building info sheet, showing you which items have been attached and which items are still pending. For a pending item, you can see its position in the fulfillment queue. You need to manufacture these items for them to be attached to the building. If there is a particular building that you need built ASAP, you can click on the "make top priority" button (or hit `CtrlT`) to bump the items for this building to the front of their respective queues.

Note that each item type and filter configuration has its own queue, so even if an item is in queue position 1, there may be other queues that snag the needed item first.

Lever linking

When linking levers, *buildingplan* extends the vanilla panel by offering control over which mechanisms are chosen for installation at the lever and at the target. Heat safety filters are provided for convenience.

Mechanism unlinking

When selecting a building linked with mechanisms, buttons to `Unlink` appear by each linked building on the `Show linked buildings` tab. This will undo the link without having to deconstruct and rebuild the target building. The unlinked mechanisms will remain a part of their respective buildings (providing value as usual) unless freed via the `Free` buttons on the `Show items` tab on both buildings. This will remove the mechanism from the building and drop it onto the ground, allowing it to be reused elsewhere. There is an option to auto-free mechanisms when unlinking to perform this step automatically.

5.6.45 burial

Tags: fort | productivity | buildings

Command: `burial`

Create tomb zones for unzoned coffins.

Creates a 1x1 tomb zone for each built coffin that isn't already contained in a zone.

Usage

```
burial [<options>]
```

Examples

`burial`
Create a general use tomb for every unzoned coffin on the map.

`burial -z`
Create tombs only on the current zlevel.

`burial -c`
Create tombs designated for burial of citizens only.

`burial -p`
Create tombs designated for burial of pets only.

`burial -cp`
Create tombs with automatic burial disabled for both citizens and pets, requiring manual assignment of deceased units to each tomb.

Options

`-z, --cur-zlevel`
Only create tombs on the current zlevel.

`-c, --citizens-only`
Only automatically bury citizens.

`-p, --pets-only`
Only automatically bury pets.

5.6.46 burrow

Tags: fort | auto | design | productivity | units

Command: `burrow`

Quickly adjust burrow tiles and units.

This tool has two modes. When enabled, it monitors burrows with names that end in +. If a wall at the edge of such a burrow is dug out, the burrow will be automatically extended to include the newly-revealed adjacent walls. If a miner digs into an open space, such as a cavern, the open space will *not* be included in the burrow.

When run as a command, it can quickly adjust which tiles and/or units are associated with the burrow.

Usage

```
enable burrow
burrow tiles|units clear <target burrow> [<target burrow> ...] [<options>]
burrow tiles|units set|add|remove <target burrow> <burrow> [...] [<options>]
burrow tiles box-add|box-remove <target burrow> [<pos>] [<pos>] [<options>]
burrow tiles flood-add|flood-remove <target burrow> [<options>]
```

The burrows can be referenced by name or by the internal numeric burrow ID. If referenced by name, the first burrow that matches the name (case sensitive) will be targeted. If a burrow name ends in + (to indicate that it should be auto-expanded), the final + does not need to be specified on the commandline.

For `set`, `add`, or `remove` commands, instead of a burrow, you can specify one of the following all-caps keywords:

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

to add or remove tiles with the corresponding properties.

Flood fill selects tiles spreading out from a starting tile if they:

- match the inside/outside and hidden/revealed properties of the starting tile
- match the walkability group of the starting tile OR (if the starting tile is walkable) is adjacent to a tile with the same walkability group as the starting tile

When flood adding, the flood fill will also stop at any tiles that have already been added to the burrow. Similarly for flood removing, the flood will also stop at tiles that are not in the burrow.

Examples

`enable burrow`
: Start monitoring burrows that have names ending in '+' and automatically expand them when walls that border the burrows are dug out.

`burrow tiles clear Safety`
: Remove all tiles from the burrow named `Safety` (in preparation for adding new tiles elsewhere, presumably).

`burrow units clear Farmhouse Workshops`
: Remove all units from the burrows named `Farmhouse` and `Workshops`.

`multicmd burrow tiles set Inside INSIDE; burrow tiles remove Inside HIDDEN`
: Reset the burrow named `Inside` to include all the currently revealed, interior tiles.

`burrow units set "Core Fort" Peasants Skilled`
: Clear all units from the burrow named `Core Fort`, then add units currently assigned to the `Peasants` and `Skilled` burrows.

burrow tiles box-add Safety 0,0,0
Add all tiles to the burrow named `Safety` that are within the volume of the box starting at coordinate 0, 0, 0 (the upper left corner of the bottom level) and ending at the current location of the keyboard cursor.

burrow tiles flood-add Safety --cur-zlevel
Flood-add the tiles on the current z-level with the same properties as the tile under the keyboard cursor to the burrow named `Safety`.

Options

-c, --cursor <pos>
Indicate the starting position of the box or flood fill. If not specified, the position of the keyboard cursor is used.

-z, --cur-zlevel
Restricts a flood fill operation to the currently visible z-level.

Note

If you are auto-expanding a burrow (whose name ends in a +) and the miner who is digging to expand the burrow is assigned to that burrow, then 1-wide corridors that expand the burrow will have very slow progress. This is because the burrow is expanded to include the next dig job only after the miner has chosen a next tile to dig, which may be far away. 2-wide cooridors are much more efficient when expanding a burrow since the "next" tile to dig will still be nearby.

Overlay

When painting burrows in the vanilla UI, a few extra mouse operations are supported. If you box select across multiple z-levels, you will be able to select the entire volume instead of just the selected area on the z-level that you are currently looking at.

In addition, you can enable a mode where double-clicking will flood fill from a target tile. If 2D flood fill is enabled, then the flood fill will not extend beyond the current z-level. It will stop at walls, forbidden doors, closed gates, and boundaries between inside and outside. When the flood hits a wall boundary, it includes one tile of wall so that units inside the burrow can interact with the walls (e.g. dig, smooth, or engrave).

If 3D flood fill is enabled, then the flood will follow stairways and ramps – anywhere a unit standing on the target tile can path to. If you start a flood fill in mid-air, then the flood will extend to other adjacent air tiles.

The box and flood fill actions respect the UI setting for whether the burrow is being added to or erased.

5.6.47 cannibalism

Tags: unavailable

Command: `cannibalism`

Allows an adventurer 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
```

5.6.48 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.

Also see *force* for creating caravans.

Usage

```
caravan [list]
caravan extend [<days> [<ids>]]
caravan happy [<ids>]
caravan leave [<ids>]
caravan unload
```

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

Examples

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

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

`caravan extend 30 0 1`
Extend the time that caravans 0 and 1 stay at the depot by 30 days. If the caravans have already started leaving, they will return to the depot.

`caravan happy`
Make the active caravans willing to trade again (after seizing goods, annoying merchants, etc.). If the caravan has already started leaving in a huff, they will return to the depot.

`caravan leave`
Makes caravans pack up and leave immediately.

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

Overlays

Additional functionality is provided on the various trade-related screens via *overlay* widgets.

Trade screen

- **`Shift+Click checkbox`: Select all items inside a bin without selecting the** bin itself
- **`Ctrl+Click checkbox`: Collapse or expand a single bin (as is possible in** the "Move goods to/from depot" screen)
- **`Ctrl+c`: Collapses all bins. The hotkey hint can also be clicked as though** it were a button.
- **`Ctrl+x`: Collapses everything (all item categories and anything** collapsible within each category). The hotkey hint can also be clicked as though it were a button.

There is also a reminder of the fast scroll functionality provided by the vanilla game when you hold shift while scrolling (this works everywhere).

You can turn the overlay on and off in *gui/control-panel*, or you can reposition it to your liking with *gui/overlay*. The overlay is named `caravan.tradeScreenExtension`.

Bring item to depot

When the trade depot is selected, a button appears to bring up the DFHack enhanced move trade goods screen. You'll get a searchable, sortable list of all your tradeable items, with hotkeys to quickly select or deselect all visible items.

There are filter sliders for selecting items of various condition levels and quality. For example, you can quickly trade all your tattered, frayed, and worn clothing by setting the condition slider to include from tattered to worn, then hitting Ctrl-V to select all.

Click on an item and shift-click on a second item to toggle all items between the two that you clicked on. If the one that you shift-clicked on was selected, the range of items will be deselected. If the one you shift-clicked on was not selected, then the range of items will be selected.

Trade agreement

A small panel is shown with a hotkey (`Ctrl-A`) for selecting all/none in the currently shown category.

Display furniture

A button is added to the screen when you are viewing display furniture (pedestals and display cases) where you can launch an item assignment GUI.

The dialog allows you to sort by name, value, or where the item is currently assigned for display.

You can search by name, and you can filter by item quality and by whether the item is forbidden.

5.6.49 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. Only adult females of the chosen race(s) will become pregnant.

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.

5.6.50 changeitem

Tags: adventure | fort | armok | items

Command: `changeitem`

Change item material, quality, and subtype.

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.

5.6.51 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.

5.6.52 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.

5.6.53 channel-safely

Tags: unavailable

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. You should also read Troubleshooting.

Features

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

risk-averse
Toggles whether to use cave-in prevention. Designations are activated in stages and their priorities along edges are modified. (default: enabled)

monitoring
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)

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)

Troubleshooting

If designations aren't switching correctly, try putting the designations into marker mode. Then press . (next) or resume. If you're debugging code you'll want these:

```
debugfilter set Info channel manager
debugfilter set Debug channel plugin
debugfilter set Trace channel group
```

5.6.54 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
```

5.6.55 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 in `dwarfmode`

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 keyboard 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.

Mud will not be cleaned out from under farm plots, since that would render the plot inoperable.

Examples

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

`clean map mud item snow`
Removes all spatter, including mud, leaves, and snow from map tiles. Farm plots will retain their mud.

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.

5.6.56 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.

`nodump`
: Cause dwarves to drop confiscated items, but don't mark them for dumping.

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.

5.6.57 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
```

5.6.58 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.

5.6.59 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
```

5.6.60 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.

5.6.61 color-schemes

Tags: unavailable

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

5.6.62 combat-harden

Tags: fort | armok | 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 option>] [<hardness option>]
```

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 citizens and residents 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.

5.6.63 combine

Tags: fort | productivity | items | plants | stockpiles

Command: `combine`

Combine items that can be stacked together.

Usage

```
combine (all|here) [<options>]
```

Examples

combine
Displays help

combine all --dry-run
Preview stack changes for all types in all stockpiles.

combine all
Merge stacks for all stockpile and all types

combine all --types=meat,plant
Merge `meat` and `plant` type stacks in all stockpiles.

combine here
Merge stacks in the selected stockpile.

Commands

all
Search all stockpiles.

here
Search the currently selected stockpile.

Options

-d, --dry-run
Display the stack changes without applying them.

-t, --types <comma separated list of types>
Filter item types. Default is `all`. Valid types are:

> `all`: all of the types listed here. `ammo`: AMMO. Qty max 25. `drink`: DRINK. Qty max 25. `fat`: GLOB and CHEESE. Qty max 5. `fish`: FISH, FISH_RAW and EGG. Qty max 5. `food`: FOOD. Qty max 20. `meat`: MEAT. Qty max 5. `parts`: CORPSEPIECE. Material max 30. `plant`: PLANT and PLANT_GROWTH. Qty max 5. `powders`: POWDERS_MISC. Qty max 10.

-q, --quiet
Only print changes instead of a summary of all processed stockpiles.

-v, --verbose n
Print verbose output, n from 1 to 4.

Notes

The following conditions prevent an item from being combined:

1. An item is not in a stockpile.
2. An item is sand or plaster.
3. An item is rotten, forbidden, marked for dumping/melting, on fire, encased, owned by a trader/hostile/dwarf or is in a spider web.
4. An item is part of a corpse and is not butchered.

The following categories are defined:

1. Corpse pieces, grouped by piece type and race
2. Items that have an associated race/caste, grouped by item type, race, and caste
3. Ammo, grouped by ammo type, material, and quality. If the ammo is a masterwork, it is also grouped by who created it.
4. Anything else, grouped by item type and material.

5.6.64 confirm

Tags: fort | interface

Command: `confirm`

Adds confirmation dialogs for destructive actions.

In the base game, it is frightenly easy to destroy hours of work with a single misclick. Now you can avoid the consequences of accidentally disbanding a squad (for example), or deleting a hauling route.

See *gui/confirm* for a configuration GUI that controls which confirmation prompts are enabled.

Usage

```
confirm [list]
confirm enable|disable all
confirm enable|disable <id> [<id> ...]
```

Run without parameters (or with the `list` option) to see the available confirmation dialogs and their IDs. You can enable or disable all dialogs or set them individually by their IDs.

5.6.65 control-panel

Tags: dfhack

Command: `control-panel`

Configure DFHack and manage active DFHack tools.

This is the commandline interface for configuring DFHack behavior, toggling which functionality is enabled right now, and setting up which tools are enabled/run when starting new fortress games. For an in-game graphical interface, please use *gui/control-panel*. For a commandline interface for configuring which overlays are enabled, please use *overlay*.

This interface controls three kinds of configuration:

1. Tools that are enabled right now. These are DFHack tools that run in the background, like *autofarm*, or tools that DFHack can run on a repeating schedule, like the "autoMilk" functionality of *workorder*. Most tools that can be enabled are saved with your fort, so you can have different tools enabled for different forts. If a tool is marked "global", however, like *hide-tutorials*, then enabling it will make it take effect for all games.

2. Tools or commands that should be auto-enabled or auto-run when you start a new fortress. In addition to tools that can be "enabled", this includes commands that you might want to run once just after you embark, such as commands to configure *autobutcher* or to drain portions of excessively deep aquifers.

3. DFHack system preferences, such as whether "Armok" (god-mode) tools are shown in DFHack lists (including the lists of commands shown by the control panel) or mouse configuration like how fast you have to click for it to count as a double click (for example, when maximizing DFHack tool windows). Preferences are "global" in that they apply to all games.

Run `control-panel list` to see the current settings and what tools and preferences are available for configuration.

Usage

```
control-panel list <search string>
control-panel enable|disable <command or number from list>
control-panel autostart|noautostart <command or number from list>
control-panel set <preference> <value>
control-panel reset <preference>
```

Examples

control-panel list butcher
Shows the current configuration of all commands related to *autobutcher* (and anything else that includes the text "butcher" in it).

control-panel enable fix/empty-wheelbarrows or control-panel enable 25
Starts to run *fix/empty-wheelbarrows* periodically to maintain the usability of your wheelbarrows. In the second version of this command, the number "25" is used as an example. You'll have to run `control-panel list` to see what number this command is actually listed as.

control-panel autostart autofarm
Configures *autofarm* to become automatically enabled when you start a new fort.

control-panel autostart fix/blood-del
Configures *fix/blood-del* to run once when you start a new fort.

control-panel set HIDE_ARMOK_TOOLS true
Enable "mortal mode" and hide "armok" tools in the DFHack UIs. Note that this will also remove some entries from the `control-panel list` output. Run `control-panel list` to see all preference options and their descriptions.

API

Other scripts can query whether a command is set for autostart via the script API:

```
local control_panel = reqscript('control-panel')
local enabled, default = control_panel.get_autostart(command)
```

5.6.66 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.

5.6.67 cursecheck

Tags: fort | armok | inspection | units

Command: `cursecheck`

Check for cursed creatures.

This command checks a single unit or the whole map for curses (ghosts, vampires, necromancers, werebeasts, zombies, etc.).

If a unit is selected, only the selected unit will be checked. Otherwise, all units on the 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).

5.6.68 cxxrandom

Tags: dev

Provides a Lua API for random distributions.

See *cxxrandom* for details.

5.6.69 deathcause

Tags: fort | inspection | units

Command: `deathcause`

Find out the cause of death for a creature.

Select a corpse or body part on the ground, and `deathcause` will detail the cause of death of the creature.

Usage

```
deathcause
```

5.6.70 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.

5.6.71 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.

5.6.72 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
```

5.6.73 design

Tags: fort | design | dev | map

Draws designations in shapes.

This plugin provides a Lua API, but no direct commands. See *gui/design* for the user interface.

5.6.74 deteriorate

Tags: unavailable

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] [--keep-usable]`
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] [--keep-usable]`
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, keeping usable parts such as hair, wool:

```
deteriorate start --types corpses --keep-usable
```

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.

-k, --keep-usable
Keep usable body parts such as hair, wool, hooves, bones, and skulls.

-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 resident corpses and body parts.

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

5.6.75 die

Tags: dfhack

Command: `die`

Instantly exit DF without saving.

Use to exit DF quickly and safely.

Usage

```
die
```

5.6.76 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>] [--zup|-u] [--zdown|-zu] [--cur-zlevel|-z] [--hidden|-h] [--no-auto|-a]
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. By default, only designates visible tiles, and in the case of dig designation, applies automatic mining to them (designates uncovered neighbouring tiles of the same type to be dug).

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.

Other options:

-d, --zdown
Only designates tiles on the cursor's z-level and below.

-u, --zup
Only designates tiles on the cursor's z-level and above.

-z, --cur-zlevel
Only designates tiles on the same z-level as the cursor.

-h, --hidden
Allows designation of hidden tiles, and picking a hidden tile as the target type.

-a, --no-auto
No automatic mining mode designation - useful if you want to avoid dwarves digging where you don't want them.

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.

Overlay

This tool also provides three overlays that are managed by the *overlay* framework.

asciicarve

The `dig.asciicarve` overlay makes carving designations visible in ASCII mode. It highlights tiles that are designated for smoothing, engraving, track carving, or fortification carving. The designations blink (slowly) so you can still see what is underneath them.

Due to the limitations of the ASCII mode screen buffer, the designation highlights may show through other interface elements that overlap the designated area.

warmdamptoolbar

The `dig.warmdamptoolbar` overlay adds a button to the toolbar at the bottom of the screen when mining mode is active. It allows you to turn on warm or damp dig mode. Tiles designated for digging while warm and/or damp dig mode is enabled will be marked with a special symbol in graphics mode or color pattern in ASCII mode. The digging designations for these tiles are protected from cancellation due to warmth or dampness (respectively). This is very useful when digging beneath a lake or just above a magma flow.

If you also have one of the vanilla autodig modes enabled, the warm/damp dig marker will be propagated along with the autodig designation. This allows you to, for example, easily autodig a mineral vein that goes through a light aquifer.

If you have already designated a z-level when you realize you need warm or damp dig protection (e.g. you have run into a light aquifer and want to continue digging), the toolbar button gives you a shortcut to add the warm or damp dig marker to all designated tiles on the current z-level. Note that it only affects tiles that are not yet revealed since revealed tiles don't benefit from the warm or damp dig designations.

Click on the new mining toolbar icon or hit `CtrlD` to bring up the configuration submenu.

warmdamp

The `dig.warmdamp` overlay makes a number of tile properties visible when in mining or smoothing mode:

- In ASCII mode, it highlights warm tiles red and damp tiles in light blue. Box selection characters and the keyboard cursor will also change color as appropriate when over a warm or damp tile. These can show through UI elements that happen to overlap the highlighted areas, just like the `asciicarve` overlay.
- The tiles marked with warm and/or damp dig from the `warmdamptoolbar` get badges (in graphics mode) or distinctive color patterns (in ASCII mode) showing their status.
- In graphics mode, the "water drop" signifying a damp tile will no longer disappear when the tile is being box selected for applying a designation.
- Aquifer tiles are shown with icons distinct from "just damp" tiles. In graphics mode, light aquifer tiles have a "two drip" icon and heavy aquifer tiles have three drips. In ASCII mode, light aquifer tiles blink slowly in blue and heavy aquifer tiles blink faster in blue.
- The warm/damp/aquifer status will be shown for tiles that are visible from the bottom. For example, if you dig out the layer underneath an aquifer and notice dripping, entering mining mode and looking at the tiles above will show their status, even if the tiles are otherwise unrevealed. This feature was added with the rationale that if the dwarves can see the effects of the tiles above them, the player should be able to as well.

5.6.77 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.

5.6.78 digFlood

Tags: unavailable

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.

5.6.79 diggingInvaders

Tags: unavailable

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.

5.6.80 diplomacy

Tags: fort | armok | inspection | military

Command: `diplomacy`

View or alter diplomatic relationships.

This tool can report on or modify the diplomatic relationships (i.e. war vs. peace) you have with other contacted civilizations. Note that a civilization is only at peace if **both** you are at peace with them **and** they are at peace with you.

Usage

```
diplomacy
diplomacy all <RELATIONSHIP>
diplomacy <CIV_ID> <RELATIONSHIP>
```

Examples

`diplomacy`
See current diplomatic relationships between you and all other contacted civs.

`diplomacy 224 peace`
Changes both your stance towards civilization 224 and their stance towards you to peace.

`diplomacy all war`
Induce the entire world to declare war on your civilization.

5.6.81 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> ...]
```

5.6.82 do-job-now

Tags: unavailable

Command: `do-job-now`

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

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
```

5.6.83 dwarf-op

Tags: unavailable

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
```

5.6.84 dwarfmonitor

Tags: unavailable

Command: `dwarfmonitor`

Report on dwarf preferences and efficiency.

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
}
```

5.6.85 dwarfvet

Tags: fort | gameplay | animals

Command: `dwarfvet`

Allow animals to be treated at hospitals.

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

If an injured animal is assigned to a pasture, it will be temporarily unassigned from the pasture while in treatment. The animal will be reassigned to its original pasture shortly after recovery.

Animals that are on restraints or in cages will not be designated for treatment.

You can enable `dwarfvet` in *gui/control-panel*, and you can choose to start `dwarfvet` automatically in new forts in the `Gameplay -> Autostart` tab.

Usage

```
enable dwarfvet
dwarfvet [status]
dwarfvet now
```

Examples

`dwarfvet`
Report on how many animals are being treated and how many are in need of treatment.

`dwarfvet now`
Assign injured animals to a free floor spot in a nearby hospital, regardless of whether the plugin is enabled.

5.6.86 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.

5.6.87 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.

5.6.88 embark-assistant

Tags: unavailable

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.

5.6.89 embark-skills

Tags: embark | 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.

5.6.90 emigration

Tags: fort | 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

```
enable emigration
```

5.6.91 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, pot, wineskin, quiver, etc.) onto the floor, allowing you to access individual items that might otherwise be hard to get to.

Note that if there are liquides in the container, they will empty onto the floor and become unusable.

Usage

```
empty-bin
```

5.6.92 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.

If you are a script developer, see *Enabling and disabling scripts* for how to expose whether your script is currently enabled or disabled.

Usage

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

Examples

`enable manipulator`
: Enable the `manipulator` plugin.

`enable manipulator search`
: Enable multiple plugins at once.

5.6.93 eventful

Tags: dev

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

See *eventful* for details.

5.6.94 exportlegends

Tags: legends | inspection

Command: `exportlegends`

Exports extended legends data for external viewing.

When run from the legends mode screen, this tool will export detailed data about your world so that it can be browsed with external programs like Legends Browser. The data is more detailed than what you can get with vanilla export functionality, and many external tools depend on this extra information.

By default, `exportlegends` hooks into the standard vanilla `Export XML` button and runs in the background when you click it, allowing both the vanilla export and the extended data export to execute simultaneously. You can continue to browse legends mode via the vanilla UI while the export is running.

To use:

- Enter legends by "Starting a new game" in an existing world and selecting Legends mode
- Ensure the toggle for "Also export extended legends data" is on (which is the default)
- Click the "Export XML" button to generate both the standard export and the extended data export

You can also generate just the extended data export by manually running the `exportlegends` command while legends mode is open.

Usage

```
exportlegends
```

Overlay

This script also provides an overlay that is managed by the *overlay* framework. When the overlay is enabled, a toggle for exporting extended legends data will appear below the vanilla "Export XML" button. If the toggle is enabled when the "Export XML" button is clicked, then `exportlegends` will run alongside the vanilla data export.

While the extended data is being exported, a status line will appear in place of the toggle, reporting the current export target and the overall percent complete.

There is an additional overlay that masks out the "Done" button while the extended export is running. This prevents the player from exiting legends mode before the export is complete.

5.6.95 exterminate

Tags: fort | armok | units

Command: `exterminate`

Kill things.

Kills any unit, or all undead, or all units of a given race. You can target any unit on a revealed tile of the map, including hidden ambushers, but caged or chained creatures cannot be killed with this tool.

Usage

```
exterminate
exterminate this [<options>]
exterminate undead [<options>]
exterminate <race>[:<caste>] [<options>]
```

Race and caste names are case insensitive.

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 goblin --method magma --only-visible
Kill all visible, hostile goblins on the map by boiling them in magma.

Options

-m, --method <method>
Specifies the "method" of killing units. See below for details.

-o, --only-visible
Specifies the tool should only kill units visible to the player. on the map.

-f, --include-friendly
Specifies the tool should also kill units friendly to the player.

Methods

exterminate can kill units using any of the following methods:

instant
Kill by blood loss, and if this is ineffective, then kill by vaporization (default).

vaporize
Make the unit disappear in a puff of smoke. Note that units killed this way will not leave a corpse behind, but any items they were carrying will still drop.

disintegrate
Vaporize the unit and destroy any items they were carrying.

drown
Drown the unit in water.

magma
Boil the unit in magma (not recommended for magma-safe creatures).

butcher
Will mark the units for butchering instead of killing them. This is more useful for pets than armed enemies.

Technical details

This tool kills by setting a unit's `blood_count` 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`, allowing the unit to vanish in a puff of smoke if the blood loss doesn't kill them.

If the method of choice involves liquids, the tile is filled with a liquid level of 7 every tick. If the target unit moves, the liquid moves along with it, leaving the vacated tiles clean.

5.6.96 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.

Select a target in the UI or enable the keyboard cursor place it over the target unit, building, or tile before running the script.

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

Usage

extinguish
Put out the selected fire.

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.

5.6.97 fastdwarf

Tags: fort | armok | units

Command: `fastdwarf`

Citizens walk fast and and finish jobs instantly.

Usage

```
enable fastdwarf
fastdwarf [status]
fastdwarf <fast mode> [<tele mode>]
```

Examples

enable fastdwarf or fastdwarf 1
Make all your citizens move and work at maximum speed.

fastdwarf
Print out current configuration.

fastdwarf 1 1
In addition to working at maximum speed, dwarves also teleport to their destinations. It is possible that units end up in places they shouldn't be, and you may need to *teleport* stranded units back to safety.

Options

Fast modes:

0
: Citizens move and work at normal rates.

1
: Citizens move and work at maximum speed.

2
: ALL units move (and work) at maximum speed, including creatures, visitors, long-term residents, and hostiles.

Tele modes:

0
: No teleportation.

1
: Citizens teleport to their job destinations.

Note that a dwarf will only teleport when:

- They are not pushing anything (like a wheelbarrow)
- They are not dragging anything/anyone or being dragged
- They are not following anyone or being followed
- They have a job with a valid destination

So you may still see dwarves walking normally or just standing still if they do not have a job to teleport to. Since jobs get done so much faster, you will likely see groups of dwarves just standing around, with individual dwarves periodically disappaearing and reappearing moments later when they have a job to do.

5.6.98 faststart

Tags: dfhack | interface

Makes the main menu appear sooner.

This plugin accelerates the initial "Loading…" screen that appears when the game first starts, so you don't have to wait as long before the Main Menu appears and you can start playing.

Usage

```
enable faststart
```

5.6.99 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.

5.6.100 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 citizens and residents focused and unstressed.

5.6.101 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.

5.6.102 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
```

5.6.103 fix-ster

Tags: unavailable

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.

5.6.104 fix-unit-occupancy

Tags: unavailable

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.

5.6.105 fixnaked

Tags: unavailable

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
```

5.6.106 fixveins

Tags: unavailable

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
```

5.6.107 flashstep

Tags: adventure | armok | units

Command: `flashstep`

Teleport your adventurer to the mouse cursor.

Keybinding: CtrlT in `dungeonmode/Default`

`flashstep` is a hotkey-friendly teleport that places your adventurer where your mouse cursor is. A small area around the target tile is revealed. If you take a step, then the normal vision cone area around the unit is revealed.

Usage

```
flashstep
```

5.6.108 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
```

5.6.109 follow

Tags: unavailable

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
```

5.6.110 forbid

Tags: fort | productivity | items

Command: `forbid`

Forbid and list forbidden items on the map.

List forbidden items, forbid all items on the map, or forbid unreachable items. Forbidding unreachable items checks if any of your citizens can walk to the item.

Usage

```
forbid
forbid all
forbid unreachable
```

5.6.111 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`
- `Megabeast`
- `WildlifeCurious`
- `WildlifeMischievous`
- `WildlifeFlier`
- `NightCreature`

5.6.112 forceequip

Tags: unavailable

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.

5.6.113 forget-dead-body

Tags: unavailable

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.

5.6.114 forum-dwarves

Tags: unavailable

Command: `forum-dwarves`

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

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
```

5.6.115 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
```

5.6.116 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 and residents 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!

Notes

If you have to repeatedly use *full-heal* on a dwarf only to have that dwarf's syndrome return seconds later, then it's likely because said dwarf still has a syndrome-causing residue on their body. To deal with this, either use `clean units` to decontaminate the dwarf or let a hospital worker wash the residue off the dwarf and THEN do a *full-heal*. Syndromes like Beast Sickness and Demon Sickness can by VERY NASTY, causing maladies like tissue necrosis.

5.6.117 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 and residents.

Target options

--all
Selects every creature on the map.

--citizens
Selects fort citizens and residents.

--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.

5.6.118 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.

5.6.119 generated-creature-renamer

Tags: unavailable

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.

5.6.120 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.

5.6.121 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
```

5.6.122 growcrops

Tags: unavailable

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.

5.6.123 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`.

5.6.124 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
hermit [status]
```

5.6.125 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.

5.6.126 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
```

5.6.127 hide-interface

Tags: interface

Command: `hide-interface`

Hide the interface layer.

This tool simply hides the interface layer so you can view the map unhindered or take clean fullscreen screenshots. The interface will remain hidden until you hit `Esc` or right click. You can still pause/unpause and move the map around, so this is an excellent way to sit back and observe fortress life a little without being distracted by announcements or statistics.

Note that the interface layer and the map layer cannot be separated in ASCII mode. This script can only hide the interface in graphics mode.

Usage

```
hide-interface
```

5.6.128 hide-tutorials

Tags: fort | interface

Command: `hide-tutorials`

Hide new fort tutorial popups.

If you've played the game before and don't need to see the tutorial popups that show up on every new fort, `hide-tutorials` can hide them for you. You can enable this tool as a system service in the "Services" tab of *gui/control-panel* so it takes effect for all new or loaded forts.

Specifically, this tool hides:

- The popup displayed when creating a new world
- The "Do you want to start a tutorial embark" popup
- Popups displayed the first time you open the labor, burrows, justice, and other similar screens in a new fort

Note that only unsolicited tutorial popups are hidden. If you directly request a tutorial page from the help, then it will still function normally.

Usage

```
enable hide-tutorials
hide-tutorials
```

If you haven't enabled the tool, but you run the command while a fort is loaded, all future popups for the loaded fort will be hidden.

5.6.129 hotkey-notes

Tags: unavailable

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
```

5.6.130 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 appears as a DFHack logo in the upper-left corner of the screen. You can bring up the menu by clicking on the logo or by hitting the global `CtrlShiftC` hotkey. You 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`.

The menu closes automatically when an action is taken or when you click or right click anywhere else on the screen.

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 shift 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.

5.6.131 infiniteSky

Tags: unavailable

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.

5.6.132 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
```

5.6.133 instruments

Tags: fort | inspection | workorders

Command: `instruments`

Show how to craft instruments or create work orders for them.

This tool is used to query information about instruments or to create work orders for them.

The `list` subcommand provides information on how to craft the instruments used by the player civilization. For single-piece instruments, it shows the skill and material needed to craft it. For multi-piece instruments, it displays the skill used in its assembly as well as information on how to craft the necessary pieces. It also shows whether the instrument is handheld or placed as a building.

The `order` subcommand is used to create work orders for an instrument and all of it's parts. The final assemble instrument -order waits for the part orders to complete before starting.

Usage

```
instruments [list]
instruments order <instrument_name> [<amount>]
```

When ordering, the default is to order one of the specified instrument (including all of its components).

Examples

`instruments`
List instruments and their recipes.

`instruments order givel 10`
If the instrument named `givel` in your world has four components, this will create a total of 5 work orders: one for assembling 10 givels, and an order of 10 for each of the givel's parts. Instruments are randomly generated, so your givel components may vary.

`instruments order ilul`
Creates work orders to assemble one ïlul. Spelling doesn't need to include the special ï character.

5.6.134 isoworldremote

Tags: unavailable

Provides a remote API used by Isoworld.

See *DFHack remote interface* for related remote APIs.

5.6.135 item

Tags: fort | productivity | items

Command: `item`

Perform bulk operations on groups of items.

Filter items in you fort by various properties (e.g., item type, material, wear-level, quality, ...), and perform bulk operations like forbid, dump, melt, and their inverses. By default, the tool does not consider artifacts and owned items. Outputs the number of items that matched the filters and were modified.

Usage

```
item [count|[un]forbid|[un]dump|[un]hide|[un]melt] [<filter options>] [<search>]
```

The `count` action counts up the items that are matched by the given filter options. Otherwise, the named property is set (or unset) on all the items matched by the filter options. The counts reported when you actually apply a property might differ from those reported by `count`, because applying a property skips over all items that already have the property set (see `--dry-run`)

The (optional) search string will filter by the item description. It will be interpreted as a Lua pattern, so any special regular expression characters (like `-`) will need to be escaped with `%` (e.g. `%-`) to be interpreted as a literal string. See https://www.lua.org/manual/5.3/manual.html#6.4.1 for details. For example "cave spider silk" will match both "cave spider silk web" and "cave spider silk cloth". Use `^pattern$` to match the entire description.

Examples

`item count steel -v`
Scan your stocks for (unowned) items that contain the word "steel" and print out the counts for each steel item found.

`item count --verbose --by-type steel`
Scan your stocks for (unowned) items that contain the word "steel" and print out the descriptions of the matched items and the counts, grouped by item type.

`item forbid --unreachable`
Forbid all items that cannot be reached by any of your citizens.

`item unforbid --inside Cavern1 --type wood`
Unforbid/reclaim all logs inside the burrow named "Cavern1" (Hint: use 3D flood-fill to create a burrow covering an entire cavern layer).

`item melt -t weapon -m steel --max-quality 3`
Designate all steel weapons whose quality is at most superior for melting.

`item hide -t boulder --scattered`
Hide all scattered boulders, i.e. those that are not in stockpiles.

`item unhide`
Makes all hidden items visible again.

Options

`-n`, `--dry-run`
Get a count of the items that would be modified by an operation, which will be the number returned by the `count` action minus the number of items with the desired property already set.

`--by-type`
Only applies to the `count` action. Outputs, in addition to the total count, a table of item counts grouped by item type.

`-a`, `--include-artifacts`
Include artifacts in the item list. Regardless of this setting, artifacts are never dumped or melted.

`--include-owned`
Include items owned by units (e.g., your dwarves or visitors)

`-i`, `--inside <burrow>`
Only include items inside the given burrow.

-o, --outside <burrow>
Only include items outside the given burrow.

-r, --reachable
Only include items reachable by one of your citizens.

-u, --unreachable
Only include items not reachable by any of your citizens.

-t, --type <string>
Filter by item type (e.g., BOULDER, CORPSE, ...). Also accepts lower case spelling (e.g. "corpse"). Use `:lua @df.item_type` to get the list of all item types.

-m, --material <string>
Filter by material the item is made out of (e.g., "iron").

-c, --mat-category <string>
Filter by material category of the material item is made out of (e.g., "metal"). Use `:lua @df.dfhack_material_category` to get a list of all material categories.

-w, --min-wear <integer>
Only include items whose wear/damage level is at least `integer`. Useful values are 0 (pristine) to 3 (XX).

-W, --max-wear <integer>
Only include items whose wear/damage level is at most `integer`. Useful values are 0 (pristine) to 3 (XX).

-q, --min-quality <integer>
Only include items whose quality level is at least `integer`. Useful values are 0 (ordinary) to 5 (masterwork). Use `:lua @df.item_quality` to get the mapping between numbers and adjectives.

-Q, --max-quality <integer>
Only include items whose quality level is at most `integer`. Useful values are 0 (ordinary) to 5 (masterwork).

--stockpiled
Only include items that are in stockpiles. Does not include empty bins, barrels, and wheelbarrows assigned as storage and transport for stockpiles.

--scattered
Opposite of `--stockpiled`

--marked=<flag>,<flag>,...
Only include items that have all provided flag set to true. Valid flags are: `forbid` (or `forbidden`), `dump`, `hidden`, `melt`, and `owned`.

--not-marked=<flag>,<flag>,...
Only include items that have all provided flag set to false. Valid flags the same as for `--marked`.

--visible
Same as `--not-marked=hidden`

-v, --verbose
Print out a description of each matched item.

API

The item script can be called programmatically by other scripts, either via the commandline interface with `dfhack.run_script()` or via the API functions defined in item.lua, available from the return value of `reqscript('item')`:

- `execute(action, conditions, options [, return_items])`

Performs `action` (`forbid`, `melt`, etc.) on all items satisfying `conditions` (a table containing functions from item to boolean). `options` is a table containing the boolean flags `artifact`, `dryrun`, `bytype`, and `owned` which correspond to the (filter) options described above.

The function `execute` performs no output, but returns three values:

1. the number of matching items
2. a table containing all matched items, if `return_items` is provided and true.
3. a table containing a mapping from numeric item types to their occurrence count, if `options.bytype=true`

- `executeWithPrinting(action, conditions, options)`

Performs the same action as `execute` and performs the same output as the `item` tool, but returns nothing.

The API provides a number of helper functions to aid in the construction of the filter table. The first argument `tab` is always the table to which the filter should be added. The final `negate` argument is optional, passing `{ negate = true }` negates the added filter condition. Below, only the positive version of the filter is described.

- **`condition_burrow(tab, burrow, negate)`**
  Corresponds to `--inside`. The `burrow` argument must be a burrow object, not a string.
- **`condition_type(tab, match, negate)`**
  If `match` is a string, this corresponds to `--type <match>`. Also accepts numbers, matching against `item:getType()`.
- **`condition_reachable(tab, negate)`**
  Corresponds to `--reachable`.
- **`condition_description(tab, pattern, negate)`**
  Corresponds to the search string passed on the commandline.
- **`condition_material(tab, match, negate)`**
  Corresponds to `--material <match>`.
- **`condition_matcat(tab, match, negate)`**
  Corresponds to `--mat-category <match>`.
- **`condition_wear(tab, lower, upper, negate)`**
  Selects items with wear level between `lower` and `upper` (Range 0-3, see above).
- **`condition_quality(tab, lower, upper, negate)`**
  Selects items with quality between `lower` and `upper` (Range 0-5, see above).
- **`condition_stockpiled(tab, negate)`**
  Corresponds to `--stockpiled`.
- **`condition_[forbid|melt|dump|hidden|owned](tab, negate)`**
  Selects items with the respective flag set to `true` (e.g., `condition_forbid` checks for `item.flags.forbid`).

  API usage example:

```
local itemtools = reqscript('item')
local cond = {}

itemtools.condition_type(cond, "BOULDER")
itemtools.execute('unhide', cond, {}) -- reveal all boulders

itemtools.condition_stockpiled(cond, { negate = true })
itemtools.execute('hide', cond, {})   -- hide all boulders not in stockpiles
```

5.6.136 jobutils

Tags: unavailable

Provides commands for interacting with jobs.

Command: `job`

Inspect or modify details of workshop jobs.

Command: `job-duplicate`

Duplicates the highlighted job.

Command: `job-material`

Alters the material of the selected job.

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.

5.6.137 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). You can also represent mouse buttons beyond the first three with `MOUSE4` through `MOUSE15`.

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.

DFHack commands can advertise the contexts in which they can be usefully run. For example, a command that acts on a selected unit can tell *keybinding* that it is not "applicable" in the current context if a unit is not actively selected.

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

Bind Ctrl-Shift-C to run the *hotkeys* command on any screen at any time:

```
keybinding add Ctrl-Shift-C hotkeys
```

Bind Ctrl-M to run *gui/mass-remove*, but only when on the main map with nothing else selected:

```
keybinding add Ctrl-M@dwarfmode/Default gui/mass-remove
```

Bind the fourth mouse button to launch *gui/teleport* when a unit is selected or *gui/autodump* when an item is selected:

```
keybinding add MOUSE4@dwarfmode/ViewSheets/UNIT gui/teleport
keybinding add MOUSE4@dwarfmode/ViewSheets/ITEM gui/autodump
```

Bind Shift + the fifth mouse button to toggle the keyboard cursor in fort or adventure mode:

```
keybinding add Shift-MOUSE5@dwarfmode|dungeonmode toggle-kbd-cursor
```

5.6.138 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.

5.6.139 labormanager

Tags: unavailable

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.

5.6.140 lair

Tags: fort | armok | map

Command: `lair`

Prevent item scatter when a site is reclaimed or visited.

This tools sets map properties to avoid item scatter when the fortress is abandoned and reclaimed or visited in adventure mode.

It is called "lair" because monster lairs have similar map properties set, but this tool does not mark the site as an actual monster lair. The representation of the site on the world map won't change. Additionally, you can't restore the state of real monster lairs using this tool.

Usage

`lair`
: Mark the map as a monster lair to prevent item scatter.

`lair reset`
: Mark the map as ordinary (not lair).

5.6.141 launch

Tags: unavailable

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
```

5.6.142 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 ID, name, activation state, and what they are linked to (if anything).

`lever pull [<options>]`
Queue a job so a dwarf will pull the lever. This is the same as selecting the lever and triggering a pull job in the UI.

`lever show`
If a lever is selected, print information about it.

`lever show --id <id>`
Center the display on the specified lever and print information about it.

Examples

`lever pull --id 42 --priority`
Queue a job to pull lever 42 at high priority.

`lever pull --id 42 --instant`
Skip the job and pull the lever with the hand of Armok!

Options

`--id`
Select the lever by ID to show/pull. Uses the currently selected lever if not specified.

`--priority`
Queue a job at high priority.

`--instant`
Instantly toggle the lever without a job.

5.6.143 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 active map tiles, leaving the rest of the world unchanged.

For more powerful aquifer editing, please see *aquifer* and *gui/aquifer*.

Usage

```
light-aquifers-only
```

If you don't ever want to have to deal with heavy aquifers, you can enable the `light-aquifers-only` command in the "Autostart" tab of *gui/control-panel* so it will be run automatically whenever you start a new fort.

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.

5.6.144 linger

Tags: unavailable

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
```

5.6.145 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)

5.6.146 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.

5.6.147 list-waves

Tags: fort | inspection | units

Command: `list-waves`

Show migration wave information.

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.

5.6.148 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.

5.6.149 load-save

Tags: unavailable

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 `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
```

5.6.150 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*.

By default, the tool only searches for visible ore veins.

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.

Options

`-a`, `--all`
: Allow undiscovered ore veins to be marked.

Examples

```
locate-ore hematite
locate-ore iron
locate-ore silver --all
```

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.

5.6.151 logistics

Tags: fort | auto | animals | items | stockpiles

Command: `logistics`

Automatically mark and route items in monitored stockpiles.

Commands act upon the stockpile selected in the UI unless another stockpile identifier is specified on the commandline.

When the plugin is enabled, it checks stockpiles marked with automelt, autotrade, autodump, and/or autotrain features twice every in-game day, and will mark valid items/animals in those stockpiles for melting, trading, dumping, and/or training, respectively.

For autotrade, items will be marked for trading only when a caravan is approaching or is already at the trade depot. Items (or bins that contain items) of which a noble has forbidden export will not be marked for trade.

Stockpiles can be registered for `logistics` features by toggling the options in the *stockpiles* overlay that comes up when you select a stockpile in the UI.

You can also use this tool to automatically assign trainers to animals that are partially trained but not yet fully domesticated. This is most useful when partially trained parents produce offspring. The offspring will inherit the training level of their parents. If a trainer is not assigned, the juveniles will eventually snap and revert to wild.

Usage

```
enable logistics
logistics [status]
logistics now
logistics add [melt] [trade] [dump] [train] [<options>]
logistics clear [all] [<options>]
logistics (enable|disable) autoretrain
```

Examples

logistics
Print a summary of all your stockpiles, their `logistics` configuration, and the number of items that are designated (or can be designated) by each of the `logistics` processors.

logistics now
Designate items in monitored stockpiles according to the current configuration. This works regardless of whether `logistics` is currently enabled.

logistics add melt
Register the currently selected stockpile for automelting. Meltable items that are brought to this stockpile will be designated for melting.

logistics add melt trade -s goblinite
Register the stockpile(s) named "goblinite" for automatic melting and automatic trading. Items will be marked for melting, but any items still in the stockpile when a caravan shows up will be brought to the trade depot for trading.

logistics clear
Unregisters the currently selected stockpile from any monitoring. Any currently designated items will remain designated.

logistics clear -s 12,15,goblinite
Unregisters the stockpiles with stockpile numbers 12 and 15, along with any stockpiles named "goblinite", from any monitoring.

logistics clear all
Unregister all stockpiles from any monitoring.

logistics enable autoretrain
Monitor animal births and automatically assign trainers to any partially trained animals.

Options

-s, --stockpile <name or number>[,<name or number>...]
Causes the command to act upon stockpiles with the given names or numbers instead of the stockpile that is currently selected in the UI. Note that the numbers are the stockpile numbers, not the building ids.

-m, --melt-masterworks
If specified with a `logistics add melt` command, will configure the stockpile to allow melting of masterworks. By default, masterworks are not marked for melting, even if they are in an automelt stockpile.

5.6.152 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.

5.6.153 lua

Tags: dfhack | dev

Command: `lua`

Run Lua script commands.

Usage

lua
Start an interactive lua interpreter. Type `quit` on an empty line and hit enter to exit the 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 (as distinct from the displayed elevation).

:lua !unit.id
Print out the id of the currently selected unit.

:lua ~item.flags
Print out the toggleable flags for the currently selected item.

:lua @df.profession
Print out the valid internal profession names.

5.6.154 luasocket

Tags: dev

Provides a Lua API for accessing network sockets.

See *luasocket* for details.

5.6.155 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>|<class>|all
Make the selected dwarf legendary in the specified skill or class of skills.

make-legendary all
Make the selected dwarf legendary in all skills.

Examples

make-legendary MINING
Make the selected dwarf a legendary miner.

make-legendary Medical
Make the selected dwarf legendary in all medical skills.

make-legendary all
Make the selected dwarf legendary in all skills. Only perfection will do.

5.6.156 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
```

5.6.157 makeown

Tags: fort | armok | units

Command: `makeown`

Converts the selected unit to be a fortress citizen.

Select a unit in the UI and run this tool to converts that unit to be a fortress citizen (if sentient). It also removes their foreign affiliation, if any.

This tool also fixes Bug 10921, where you request workers from your holdings, but they come with the "Merchant" profession and are unable to complete jobs in your fort. Select those units and run *makeown* to convert them into functional citizens.

Usage

```
makeown
```

5.6.158 manipulator

Tags: unavailable

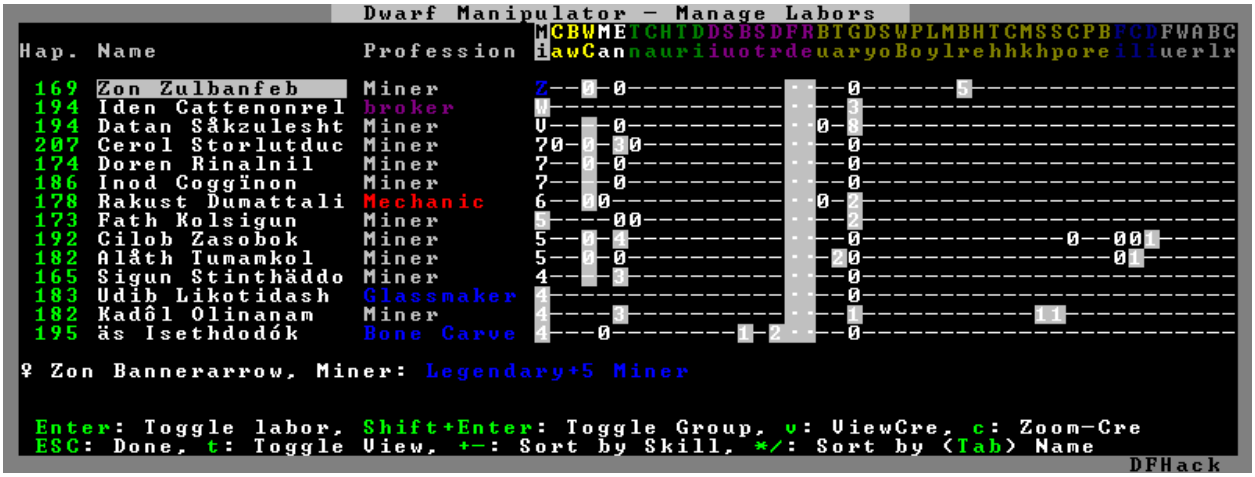
An in-game labor management interface.

It is equivalent to the popular Dwarf Therapist utility.

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

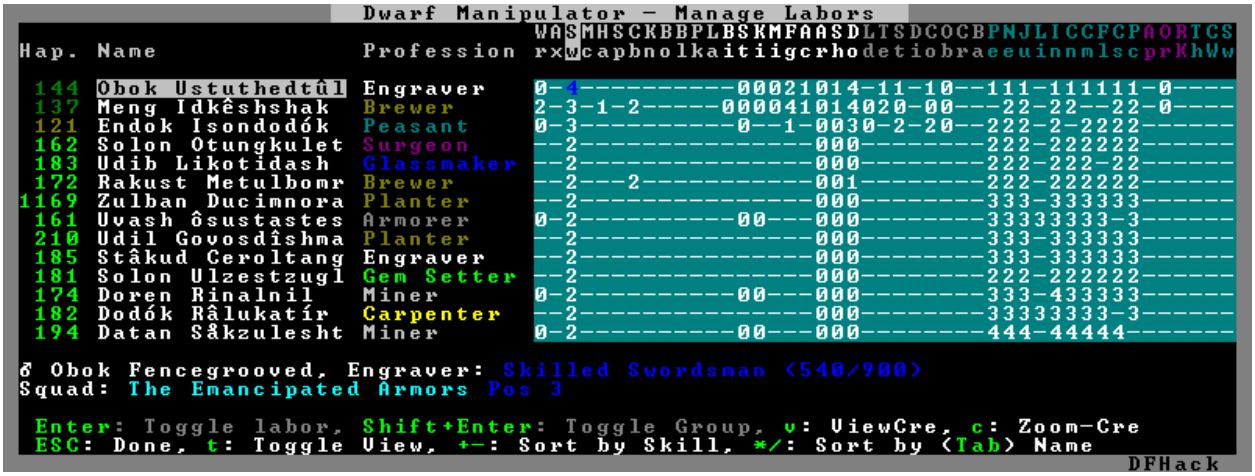
Usage

```
enable manipulator
```



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.



Press `t` to toggle between Profession, Squad, and Job views.

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

Press the Z-Up (<) and Z-Down (>) keys to move quickly between labor/skill categories. The numpad Z-Up and Z-Down keys seek to the first or last unit in the list. `Backspace` seeks to the top left corner.

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

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

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

The following mouse shortcuts are also available:

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

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

Professions

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

To save a profession, highlight a dwarf and press `P`. The profession will be saved using the custom profession name of the dwarf, or the default 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
```

5.6.159 map-render

Tags: unavailable

Provides a Lua API for re-rendering portions of the map.

See *map-render* for details.

5.6.160 markdown

Tags: adventure | fort | items | units

Command: `markdown`

Export displayed text to a Markdown file.

Keybinding: CtrlT in `dwarfmode/ViewSheets/UNIT|dwarfmode/ViewSheets/ITEM|dungeonmode/ViewSheets/UNIT|dungeonmode/ViewSheets/ITEM`

Saves the description of a selected unit or item to a Markdown file encoded in UTF-8.

By default, data is stored in the `markdown_{YourWorldName}.md` file in the root of the game directory.

For units, the script exports:

1. Name, race, age, profession
2. Description from the unit's Health -> Description screen
3. Traits from the unit's Personality -> Traits screen

For items, it exports:

1. Decorated name (e.g., "«Chalk Statue of Dakas»")
2. Full description from the item's view sheet

The script works for most items with in-game descriptions and names, including those in storage, on the ground, installed as a building, or worn/carried by units.

By default, entries are appended, not overwritten, allowing the `markdown` command to compile descriptions of multiple items & units in a single document. You can quickly export text for the currently selected unit or item by tapping the Ctrl-t keybinding or selecting *markdown* from the DFHack logo menu.

Usage

```
markdown [<name>] [<options>]
```

Specifying a name will append to `markdown_{name}.md`, which can be useful for organizing data by category or topic. If `name` includes whitespace, quote it in double quotes.

If no `name` is given, the name of the loaded world is used by default.

Examples

- `markdown`

Example output for a selected chalk statue in the world "Orid Tamun", appended to the default `markdown_Orid_Tamun.md` file:

```
[...previous entries...]

### Chalk Statue of Bìlalo Bandbeach

#### Description:
This is a well-crafted chalk statue of Bìlalo Bandbeach. The item is an
image of Bìlalo Bandbeach the elf and Lani Lyricmonks the Learned the ettin
in chalk by Domas Uthmiklikot. Lani Lyricmonks the Learned is striking down
Bìlalo Bandbeach.
The artwork relates to the killing of the elf Bìlalo Bandbeach by the
ettin Lani Lyricmonks the Learned with Hailbite in The Forest of
Indignation in 147.

---
```

- `markdown -o descriptions`

Example output for a selected unit Lokum Alnisendok, written to the newly overwritten `markdown_descriptions.md` file:

```
### Lokum Alnisendok, dwarf, 27 years old Presser.

#### Description:
A short, sturdy creature fond of drink and industry.

He is very quick to tire.

His very long beard is neatly combed.  His very long sideburns are braided.
His very long moustache is neatly combed.  His hair is clean-shaven.  He is
average in size. His nose is sharply hooked.  His nose bridge is convex.
His gold eyes are slightly wide-set.  His somewhat tall ears are somewhat
narrow.  His hair is copper.  His skin is copper.

#### Personality:
He has an amazing memory, but he has a questionable spatial sense and poor
focus.

He doesn't generally think before acting.  He feels a strong need to
```

(continues on next page)

(continued from previous page)

```
reciprocate any favor done for him.  He enjoys the company of others.  He
does not easily hate or develop negative feelings.  He generally finds
himself quite hopeful about the future.  He tends to be swayed by the
emotions of others.  He finds obligations confining, though he is
conflicted by this for more than one reason.  He doesn't tend to hold on to
grievances.  He has an active imagination.

He needs alcohol to get through the working day.

---
```

Options

-o, --overwrite
Overwrite the output file, deleting previous entries.

Setting up custom keybindings

If you want to use custom filenames, you can create your own keybinding so you don't have to type out the full command each time. You can run a command like this in *gui/launcher* to make it active for the current session, or add it to `dfhack-config/init/dfhack.init` to register it at startup for future game sessions:

```
keybinding add Ctrl-Shift-S@dwarfmode/ViewSheets/UNIT|dwarfmode/ViewSheets/ITEM
↪"markdown descriptions"
```

You can use a different key combination and output name, of course. See the *keybinding* docs for more details.

Alternately, you can register commandlines with the *gui/quickcmd* tool and run them from the popup menu.

5.6.161 max-wave

Tags: unavailable

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.

5.6.162 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
```

5.6.163 misery

Tags: fort | gameplay | units

Command: `misery`

Make citizens more miserable.

When enabled, all of your citizens receive a negative thought about a particularly nasty soapy bath. You can vary the strength of this negative thought to increase or decrease the difficulty of keeping your citizens happy.

Usage

```
enable misery
misery [status]
misery <factor>
misery clear
```

The default misery factor is 2, which will result in a moderate hit to your dwarves' happiness. Larger numbers increase the challenge.

Examples

enable misery
: Start adding bad thoughts about nasty soapy baths to your citizens!

misery 5
: Change the strength of the soapy bath negative thought to something quite large – this is very challenging to handle!

misery clear
: Clear away negative thoughts added by `misery`. Note that this will not clear negative thoughts that your dwarves accumulated "naturally".

5.6.164 mode

Tags: unavailable

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.

5.6.165 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
```

5.6.166 names

Tags: unavailable

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
```

5.6.167 necronomicon

Tags: fort | inspection | items

Command: `necronomicon`

Find books that contain the secrets of life and death.

Lists all books in the fortress that contain the secrets to life and death. To find the books in fortress mode, go to the Written content submenu in Objects (O). Slabs are not shown by default since dwarves cannot read secrets from a slab in fort mode.

Usage

```
necronomicon [<options>]
```

Options

-s, --include-slabs
: Also list slabs that contain the secrets of life and death. Note that dwarves cannot read the secrets from a slab in fort mode.

5.6.168 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
```

5.6.169 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
```

5.6.170 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.

5.6.171 open-legends

Tags: legends | inspection

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 as normal, and even export legends data while you're on the legends screen.

However, entering and leaving legends mode from a fort or adventure mode game will leave Dwarf Fortress in an inconsistent state. Therefore, entering legends mode via this tool is a **ONE WAY TRIP**. If you care about your savegame, you *MUST* save your game before entering legends mode. *open-legends* will pop up a dialog to remind you of this and will give you a link that you can use to trigger an Autosave. You can also close the dialog, do a manual save with a name of your choice, and run *open-legends* again to continue to legends mode.

Usage

```
open-legends
open-legends --no-autoquit
```

Options

The `--no-autoquit` option is provided for bypassing the auto-quit in case you are doing testing where you want to switch into legends mode, switch back, make a few changes, and then hop back into legends mode. However, please note that while the game appears playable once you are back in the original mode, your world data **is corrupted** in subtle ways that are not easy to detect from the UI. Once you are done with your legends browsing, you *must* quit to desktop and restart the game to be sure to avoid save corruption issues.

Upon return to the playable game, autosaves will be disabled to avoid accidental overwriting of good savegames.

If the `--no-autoquit` option has previously been passed and the savegame is already "tainted" by previous trips into legends mode, the warning dialog prompting you to save your game will be skipped.

5.6.172 orders

Tags: fort | productivity | workorders

Command: `orders`

Manage manager orders.

Usage

`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 recheck [this]`
Sets the status to `Checking` (from `Active`) for all work orders that have conditions that can be re-checked. If the "this" option is passed, only sets the status for the workorder whose condition details page is open. 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.

`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 `dfhack-config/init/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.

Overlay

Fort-wide work orders screen

Orders plugin functionality is directly available via an *overlay* widget when the fort-wide work orders screen is open. There are hotkeys assigned to export, import, sort, clear, and recheck conditions. You can also click on the hotkey hints as if they were buttons. Clearing will ask for confirmation before acting.

When you open the conditions screen for a manager order, there is also a small overlay that allows you to recheck conditions for just that order. This is useful for when the conditions were true when the order started, but they have become false and now you're just getting repeated cancellation spam as the order cannot be fulfilled.

Workshop Workers tab

For workshops that do *not* have a workshop master assigned, there is a slider you can use to restrict the units that perform jobs at that workshop by their skill level.

Due to space constraints, some skill levels are combined with the adjacent higher rank on the slider:

- "Competent" includes "Adequate" workers
- "Proficient" includes "Skilled" workers
- "Expert" includes "Adept" workers
- "Accomplished" includes "Professional" workers
- "Master" includes "Great" workers
- "Grand Master" includes "High Master" workers

Finally, a list is shown for workshops that service manager orders of multiple labor types. You can toggle the listed labors so the workshop only accepts general work orders that match the enabled labors (the list of allowed labors is different for every workshop).

For example, by default, all weapon, armor, and blacksmithing general manager orders get sent to all forges that can take general work orders. With labor restrictions, you can designate specific forges to handle just weapons, just armor, or just metalsmithing. Then, you can assign appropriate legendary masters to each forge, and they will only receive orders for appropriate products.

Simiarly, you can set up Craftsdwarf's workshops to specialize in stone, wood, or bone.

Regardless of the labor restriction settings, you can manually assign any task to the workshop and it will still be completed. The labor restrictions only apply to general manager work orders scheduled from the fort-wide work orders screen.

Veteran players may remember these overlays as vanilla features in pre-v50 Dwarf Fortress. This is actually still the case. The DFHack overlay simply provides a UI for the vanilla feature hiding beneath the surface.

If you want to change where the overlay panels appear, you can move them with *gui/overlay*.

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/bins/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.

Note that the jugs are specifically made out of wood. This is so, as long as you don't may any other "Tools" out of wood, you can have a stockpile just for jugs by restricting a finished goods stockpile to only take wooden tools.

Armok's additional 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/quivers/armor
- silk cloaks
- bone/wooden bolts
- smelting for platinum, silver, steel, bronze, bismuth bronze, and copper (and their dependencies)
- bronze/bismuth bronze/copper bolts
- steel/silver/iron/bismuth bronze/bronze/copper weapons and armor, with checks to ensure only the best available materials are being used
- wooden shields (if metal isn't available)

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)

5.6.173 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/control-panel*. 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

Overlay

The *overlay* plugin also provides a standard overlay itself: title_version, which displays the DFHack version on the DF title screen, along with quick links to *quickstart-guide* and *gui/control-panel*.

5.6.174 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 and *gui/pathable* for the user interface.

5.6.175 pet-uncapper

Tags: fort | gameplay | animals

Command: pet-uncapper

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 pet-uncapper
Enables the plugin and starts running with default settings.

pet-uncapper [status]
Print out current settings.

pet-uncapper now
Impregnate adult female pets that have access to a compatible male, up to the population cap.

pet-uncapper 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.

pet-uncapper every <ticks>
Set how often the plugin will cause pregnancies. The default frequency is every 10,000 ticks (a little over 8 game days).

pet-uncapper 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.

5.6.176 plants

Tags: unavailable

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.

5.6.177 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.

5.6.178 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.

5.6.179 pop-control

Tags: unavailable

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.

5.6.180 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
```

5.6.181 power-meter

Tags: unavailable

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.

5.6.182 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.
```

5.6.183 prefchange

Tags: unavailable

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

5.6.184 preserve-tombs

Tags: fort | bugfix

Command: `preserve-tombs`

Preserve tomb assignments when assigned units die.

If you find that the tombs you assign to units get unassigned from them when they die (e.g. your nobles), this tool can help fix that.

Usage

`enable preserve-tombs`
enable the plugin

`preserve-tombs [status]`
check the status of the plugin, and if the plugin is enabled, lists all currently tracked tomb assignments

`preserve-tombs now`
forces an immediate update of the tomb assignments. This plugin automatically updates the tomb assignments once every 100 ticks.

This tool runs in the background.

5.6.185 prioritize

Tags: fort | auto | jobs

Command: `prioritize`

Automatically boost the priority of important job types.

This tool encourages 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.

Usage

```
enable prioritize
disable prioritize
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. The counts are saved with your game, so they will be accurate even if the game has been saved and reloaded since `prioritize` was started.

`enable prioritize, prioritize -a defaults`
Watch for and 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.

`disable prioritize`
Remove all job types from the watch list and clear tracking data.

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 jobs of those types are listed.

-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 those logs and/or that free space. Prioritizing tree chopping will get your dwarves on the task and keep you from staring at the screen 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. It is a good idea to enable *prioritize* with at least these defaults for all your forts.

The default prioritize list includes:

- Handling items that can rot
- Medical, hygiene, and hospice tasks
- Interactions with animals and prisoners

- Noble-specific tasks (like managing workorders)
- Dumping items, felling trees, and other tasks that you, as a player, might stare at and internally scream "why why why isn't this getting done??".

5.6.186 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 [<options>]`
: Displays properties of the tile highlighted with the keyboard cursor or at the specified coordinates. If you do not have a keyboard cursor visible, enter a mode that shows the keyboard cursor (like mining mode). If the keyboard cursor is still not visible, hit Alt-k to invoke *toggle-kbd-cursor*.

`bprobe`
: Displays properties of the selected building. For deeper inspection of the building, see *gui/gm-editor*.

`cprobe`
: Displays properties of the selected unit. 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*.

Options

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

5.6.187 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 currently highlighted 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.

5.6.188 putontable

Tags: unavailable

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.

5.6.189 questport

Tags: unavailable

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
```

5.6.190 quickfort

Tags: fort | design | productivity | buildings | map | stockpiles

Command: `quickfort`

Apply layout blueprints to your fort.

Quickfort reads stored blueprint files and applies them to the game map. You can apply blueprints that designate digging, build buildings, place stockpiles, mark zones, and more. If you find yourself spending time doing similar or repetitive designs 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. Player-created blueprints are stored in the `dfhack-config/blueprints` directory.

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. Additional library blueprints can be *added with mods* as well.

Usage

quickfort list [-m|--mode <mode>] [-u|--useronly] [-h|--hidden] [<search string>]
Lists available blueprints. 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, library blueprints 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 defaults.

<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 scheduled for destruction (depending on their construction status), and stockpiles/zones are removed.

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 "notes" 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 interactively 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 keyboard map cursor for the the blueprint start position. If this option is specified, then an active keyboard 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.

-m, --marker <type>[,<type>...]
Apply the given marker(s) to the tiles designated by the `#dig` blueprint that you are applying. Valid marker types are: `blueprint` (designate but don't dig), `warm` (dig even if the tiles are warm), and `damp` (dig even if the tiles are damp). `warm` and `damp` markers are interpreted by the *dig* tool for interruption-free digging through warm and damp tiles.

-p, --priority <num>
Set the priority to the given number (1-7) for tiles designated by the `#dig` blueprint that you are applying. That is, tiles that normally have a priority of `4` will instead have the priority you specify. If the blueprint uses other explicit priorities, they will be shifted up or down accordingly.

--preserve-engravings <quality>
Don't designate tiles for digging/carving 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 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 has a few global configuration options that you can customize with the `quickfort set` command. Modified settings are only kept for the current session and will be reset when you restart DF.

blueprints_user_dir (default: dfhack-config/blueprints)
Directory tree to search for player-created blueprints. It can be set to an absolute or relative path. If set to a relative path, it 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 `dfhack-config/blueprints` dir).

blueprints_library_dir (default: hack/data/blueprints)
Directory tree to search for library blueprints.

force_marker_mode (default: false)
If true, will designate all dig blueprints in marker=blueprint mode. If false, only cells with dig codes explicitly prefixed with `mb` in the blueprint cell will be designated in marker mode.

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 assign an appropriate number of wheelbarrows to the stockpile. Blueprints can *override* this value for specific stockpiles.

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 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 instead of a table 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 blueprint alias names to their expansions. If not specified, defaults to `{}`.

`marker`
A map of strings to booleans indicating which markers should be applied to this `dig` mode blueprint. See *Command options* above for details. If not specified, defaults to `{blueprint=false, warm=false, damp=false}`.

`priority`
An integer between 1 and 7, inclusive, indicating the base priority for this `dig` blueprint. If not specified, defaults to 4.

`preserve_engravings`
Don't designate tiles for digging or carving 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 quickfort = reqscript('quickfort')

-- dig a 10x10 block at the mouse cursor position
quickfort.apply_blueprint{mode='dig', data='d(10x10)',
```

(continues on next page)

(continued from previous page)

```
                            pos=dfhack.gui.getMousePos()}

-- dig a 10x10 block starting at coordinate x=30, y=40, z=50
quickfort.apply_blueprint{mode='dig', data={[50]={[40]={[30]='d(10x10)'}}}}
```

5.6.191 quicksave

Tags: dfhack | fort

Command: `quicksave`

Immediately autosave the game.

Keybinding: CtrlAltS in `dwarfmode`

When this tool is called with a fort loaded, DF will immediately do an autosave. Note that the game only keeps the last 3 autosaves.

Usage

```
quicksave
```

5.6.192 quickstart-guide

Tags:

Command: `quickstart-guide`

In-game quickstart guide for new users of DFHack.

This command displays a window in-game where a player can browse the quickstart guide from the official DFHack docs, section by section.

Usage

```
quickstart-guide
```

5.6.193 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.

5.6.194 regrass

Tags: adventure | fort | armok | animals | map

Command: `regrass`

Regrow surface grass and cavern moss.

This command can refresh the grass (and subterranean moss) growing on your map. Operates on floors, stairs, and ramps. Also works underneath shrubs, saplings, and tree trunks. Ignores furrowed soil and wet sand (beaches).

Usage

```
regrass [<pos> [<pos>]] [<options>]
```

Regrasses the entire map by default, restricted to compatible tiles in map blocks that had grass at some point. Supplying a `pos` argument can limit operation to a single tile. Supplying both can operate on a cuboid. `pos` should normally be in the form `0,0,0`, without spaces. The string `here` can be used in place of numeric coordinates to use the position of the keyboard cursor, if active. The `--block` and `--zlevel` options use the `pos` values differently.

Options

-m, --max
: Maxes out every grass type in the tile, giving extra grazing time. Not normal DF behavior. Tile will appear to be the first type of grass present in the map block until that is depleted, moving on to the next type. When this option isn't used, non-depleted grass tiles will have their existing type refilled, while grass-depleted soils will have a type selected randomly.

-n, --new
: Adds biome-compatible grass types that were not originally present in the map block. Allows regrass to work in blocks that never had any grass to begin with. Will still fail in incompatible biomes.

-f, --force
: Force a grass type on tiles with no compatible grass types. The `--new` option takes precidence for compatible biomes, otherwise such tiles will be forced instead. By default, a single random grass type is selected from the world's raws. The `--plant` option allows a specific grass type to be specified.

-p <grass_id>, --plant <grass_id>
: Specify a grass type for the `--force` option. `grass_id` is not case-sensitive, but must be enclosed in quotes if spaces exist. Providing an empty string with "" will print all available IDs and skip regrass.

-a, --ashes
: Regrass tiles that've been burnt to ash.

-d, --buildings
: Regrass tiles under certain passable building tiles including stockpiles, planned buildings, workshops, and farms. (Farms will convert grass tiles to furrowed soil after a short while, but is useful with `--mud` option.) Doesn't work on "dynamic" buildings such as doors and floor grates. (Dynamic buildings also include levers and cage traps, for some reason.) Existing grass tiles will always be refilled, regardless of building tile.

-u, --mud
: Converts non-smoothed, mud-spattered stone into grass. Valid for layer stone, obsidian, and ore.

-b, --block
Only regrass the map block that contains the first `pos` argument. *devel/block-borders* can be used to visualize map blocks.

-z, --zlevel
Regrass entire z-levels. Will do all z-levels between `pos` arguments if both given, z-level of first `pos` if one given, else z-level of viewscreen if no `pos` given.

Examples

regrass
Regrass the entire map, refilling existing and depleted grass.

regrass here
Regrass the selected tile, refilling existing and depleted grass.

regrass here 0,0,90 --zlevel
Regrass all z-levels including the selected tile's z-level through z-level 90, refilling existing and depleted grass.

regrass 0,0,100 19,19,119 --ashes --mud
Regrass tiles in the 20x20x20 cube defined by the coords, refilling existing and depleted grass, and converting ashes and muddy stone (if respective blocks ever had grass.)

regrass 10,10,100 -baudnm
Regrass the block that contains the given coord; converting ashes, muddy stone, and tiles under buildings; adding all compatible grass types, and filling each grass type to max.

regrass -f
Regrass the entire map, refilling existing and depleted grass, else filling with a randomly selected grass type if non-existent.

regrass -p ""
Print all valid grass raw ids. Don't regrass.

regrass -zf -p underlichen
Regrass the current z-level, refilling existing and depleted grass, else filling with `underlichen` if non-existent.

regrass here -bnf -p "dog's tooth grass"
Regrass the selected block, adding all compatible grass types to block data, `dog's tooth grass` if no compatible types exist. Refill existing grass on each tile, else select one of the block's types if depleted or previously non-existent.

Troubleshooting

`debugfilter set Debug regrass log` can be used to figure out why regrass is failing on a tile. (Avoid regrassing large parts of the map with this enabled, as it will make the game unresponsive and flood the console for several minutes!)

Disable with `debugfilter set Info regrass log`.

5.6.195 rejuvenate

Tags: fort | armok | units

Command: `rejuvenate`

Resets unit age.

If your most valuable citizens are getting old, this tool can save them. It decreases the age of the selected dwarf to 20 years, or to the age specified. Age is only increased using the –force option.

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.

rejuvenate --age 149 --force
Set the age of the selected dwarf to 149, even if they are younger.

Options

--all
Rejuvenate all citizens, not just the selected one.

--age <num>
Sets the target to the age specified. If this is not set, the target age is 20.

--force
Set age for units under the specified age to the specified age. Useful if there are too many babies around…

--dry-run
Only list units that would be changed; don't actually change ages.

5.6.196 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.

5.6.197 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`).

5.6.198 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
```

5.6.199 rename

Tags: unavailable

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.

5.6.200 rendermax

Tags: unavailable

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.

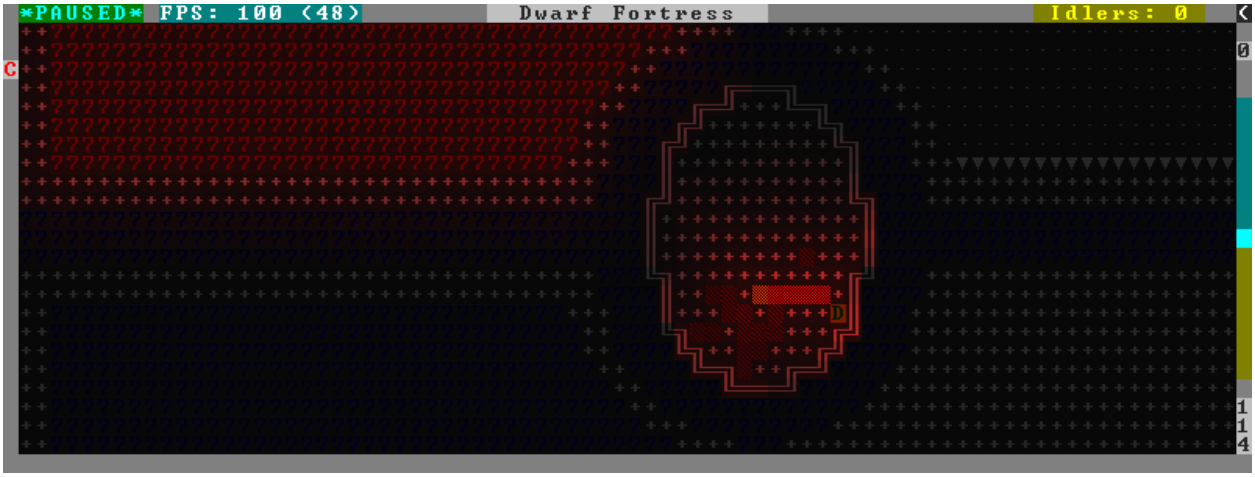
rendermax light sun <hour>|cycle
Set the outside lighting to correspond with the specified day hour (1-24), or specify `cycle` to have the lighting follow the sun (which is the default).

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.

5.6.201 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.

5.6.202 resurrect-adv

Tags: adventure | armok | units

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.

5.6.203 reveal

Tags: adventure | fort | armok | inspection | map

Command: `reveal`

Reveal the map.

Command: `unreveal`

Revert a revealed map to its unrevealed state.

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 a unit.

This reveals all z-layers in fort and adventure mode. The effect persists until you run `unreveal`.

In graphics mode, solid tiles that are not adjacent to open space will not be rendered, but they can still be examined by hovering over them with the mouse. Switching to ASCII mode (in the game settings) will allow the display of the revealed solid tiles.

For a GUI that auto-restores the map when closed, see *gui/reveal*.

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 (or treasures) from spawning. If you really want to unpause with secrets revealed, specify `demon` instead of `hell`. Note that unpausing with secrets revealed may result in a flood of announcements about the revealed secrets!

`unreveal`
Reverts the effects of `reveal` if run immediately afterwards. If you have saved the fort in a revealed state and want to restore the map, use `revflood`.

`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 keyboard cursor (if enabled) or the selected unit (if a unit is selected) or else a random citizen. 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.

Caveats

Sometimes, the map generates secret hollows adjacent to caverns in such a way that the ceiling of the hollow collapses on the first tick of the embark, leaving the hollow exposed to the caverns. In this case, the secret event will be triggered as soon as the cavern is discovered and that tile is unhidden. This would happen anyway even if you don't use *reveal*, but be aware that it is possible to trigger *some* events when you run *reveal*, even without the `hell` option.

When running `unreveal` to restore the map in adventure mode, the vision cone for the adventurer isn't fully restored until the adventurer takes a step.

5.6.204 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.

5.6.205 reveal-hidden-sites

Tags: adventure | embark | fort | 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, or on the embark map.

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

Usage

```
reveal-hidden-sites
```

5.6.206 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
```

5.6.207 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.

5.6.208 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.

5.6.209 season-palette

Tags: unavailable

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.

5.6.210 seedwatch

Tags: fort | auto | plants

Command: `seedwatch`

Manages seed and plant cooking based on seed stock levels.

Unlike brewing and other kinds of processing, cooking plants does not produce a usable seed. By default, all plants are allowed to be cooked. This often leads to the situation where dwarves have no seeds left to plant crops with because they cooked all the relevant plants. Seedwatch protects you from this problem.

Each seed type can be assigned a target stock amount. If the number of seeds of that type falls below that target, then the plants and seeds of that type will be protected from cookery. If the number rises above the target + 20, then cooking will be allowed again.

Usage

`enable seedwatch`
Start managing seed and plant cooking. By default, all types are watched with a target of `30`, but you can adjust the list or even `seedwatch clear` it and start your own if you like.

`seedwatch [status]`
Display whether seedwatch is enabled and prints out the watch list, along with the current seed counts.

`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. If you set the target to `0`, it removes the specified type from the watch list.

`seedwatch clear`
Clears all types from the watch list. Same as `seedwatch all 0`.

To see a list of all plant types that you might want to set targets for, you can run this command:

```
devel/query --table df.global.world.raws.plants.all --search ^id --maxdepth 1
```

Examples

enable seedwatch
Adds all seeds to the watch list, sets the targets to 30, and starts monitoring your seed stock levels.

seedwatch MUSHROOM_HELMET_PLUMP 50
Add Plump Helmets to the watch list and sets the target to 50.

seedwatch MUSHROOM_HELMET_PLUMP 0
removes Plump Helmets from the watch list.

5.6.211 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.

5.6.212 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
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

5.6.213 setfps

Tags: dfhack | 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>
```

5.6.214 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
```

5.6.215 showmood

Tags: fort | armok | inspection | jobs | units

Command: `showmood`

Shows all items needed for the active strange mood.

Usage

```
showmood
```

5.6.216 siege-engine

Tags: unavailable

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*.

5.6.217 siren

Tags: unavailable

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).

5.6.218 sort

Tags: fort | productivity | interface

Search and sort lists shown in the DF interface.

The `sort` tool provides search and sort functionality for lists displayed in the DF interface.

Searching and sorting functionality is provided by *overlay* widgets, and widgets for individual lists can be moved via *gui/overlay* or turned on or off via *gui/control-panel*.

Squad assignment overlay

The squad assignment overlay adds annotation, sorting, and filtering capabilities to the squad assignment list. The currently selected annotation is shown in the panel to the left of the unit list. Clicking on the arrow icon above the annotations will sort the unit list by that annotation value, or, if the list is already sorted by the annotation value, will switch between descending and ascending sort.

Annotations and sort orders

You can change which attribute is shown in the annotation panel by clicking on the `Show:` button or hitting the `S` hotkey. Alternately, if you hover the mouse over the `Show:` button, a popout panel will appear where you can select the annotation attribute directly. Selecting a new annotation attribute will also automatically sort by that attribute. You can then select one of the vanilla sorts to keep that annotation while sorting by another field.

You can choose to annotate and sort by many useful attributes, including current combat effectiveness, stress level, various military-related skills, and long-term military potential. The annotation selection and filter settings will be saved for you when you close and reopen the squad member assignment panel.

If "melee effectiveness" is selected (the default), then the citizens are annotated according to how well they will perform in battle when using the weapon they have the most skill in. The effectiveness rating also takes into account physical and mental attributes as well as general fighting (non-weapon) skills.

Simiarly, the "ranged effectiveness" annotation indicates expected effectiveness with a crossbow. This rating also takes into account relevant physical and mental attributes.

The "effectiveness" ratings are the ones you should be using if you need the best squad you can make right now. The numbers to the left of the unit list indicate exactly how effective that dwarf is expected to be. Light green numbers indicate the best of the best, while red numbers indicate dwarves that will not be effective in the military in their current state (though see "melee potential" and "ranged potential" ratings below for predictions about future effectiveness).

The "arrival order" sorts your citizens according to the most recent time they entered your map. The numbers on the left indicate the relative arrival order, and the numbers for the group of dwarves that most recently entered the map will be at the top and be colored bright green. If you run this sort after you get a new group of migrants, the migrant wave will be colored bright green. Dwarves that arrived earlier will have numbers in yellow, and your original dwarves (if any still survive and have never left and re-entered the map) will have numbers in red.

The "stress" sort order will bring your most stressed dwarves to the top, ready for addition to a therapy squad to help improve their mood.

Similarly, sorting by "need for training" will show you the dwarves that are feeling the most unfocused because they are having their military training needs unmet.

Both "stress" and "need for training" sorts use the dwarf happiness indicators to show how dire the dwarf's situation is and how much their mood might be improved if you add them to an appropriate squad.

If sorting is done by "melee potential", then citizens are arranged based on genetic predispositions in physical and mental attributes, as well as body size. Dwarves (and other humanoid creatures) with higher ratings are expected to be more effective in melee combat if they train their attributes to their genetic maximum.

Similarly, the "ranged potential" sort orders citizens by genetic predispositions in physical and mental attributes that are relevant to ranged combat. Dwarves (and other humanoid creatures) with higher rating are expected to be more effective in ranged combat if they train their attributes to the maximum.

"Melee skill effectiveness", "ranged skill effectiveness", "melee combat potential", and "ranged combat potential" are explained in detail here: https://www.reddit.com/r/dwarffortress/comments/163kczo/enhancing_military_candidate_selection_part_3/

Filters

The squad assignment panel also offers options for filtering which dwarves are shown. Each filter option can by cycled through "Include", "Only", and "Exclude" settings. "Include" does no filtering, "Only" shows only units that match the filter, and "Exclude" shows only units that do *not* match the filter.

The following filters are provided:

- Other squads: Units that are assigned to other squads
- Officials: Elected and appointed officials, like you mayor, priests, tavern keepers, etc.
- Nobility: Your monarch, barons, counts, etc.
- With infants: Mothers with infants (you may not want mothers using their babies as shields)
- Hates combat: Units that have facets and values that indicate that they will react poorly to the stresses of battle
- Maimed: Critically injured units that have lost their ability to see, grasp weapons, or walk

The "Hates combat" filter is explained in more detail here: https://www.reddit.com/r/dwarffortress/comments/1617s11/enhancing_military_candidate_selection_part_2/

Info tabs overlay

The Info overlay adds search support to many of the fort-wide "Info" panels that don't already have search widgets (e.g. "Tasks", "Artifacts", etc.). When searching for units, you can search by name (with either English or native language last names), profession, or special status (like "necromancer"). If there is text in the second column, you can search for that text as well. This is often a job name or a status, like "caged". The work animal assignment page can also filter by squad or burrow membership.

Work animals overlay

The work animal assignment screen has an overlay that annotates each visible unit with the number of work animals that unit already has assigned.

There is also a filter panel at the bottom that allows you to filter by military status, squad membership, or burrow membership (or lack thereof).

Interrogation overlay

In the interrogation and conviction screens under the "Justice" tab, you can filter by the classification of the unit. The classification groups are ordered by how likely a member of that group is to be involved in a plot. The groups are: All, Risky visitors, Other visitors, Residents, Citizens, Animals, Deceased, and Others. "Risky" visitors are those who are especially likely to be involved in plots, such as criminals, necromancers, necromancer experiments, and intelligent undead.

On the interrogations screen, you can also filter units by whether they have already been interviewed in realation to the current crime.

Candidates overlay

When you select the button to choose a candidate to assign to a noble role on the nobles screen, you can search for units by name, profession, or any of the skills in which they have achieved at least "novice" level. For example, when assigning a broker, you can search for "appraisal" to find candidates that have at least some appraisal skill.

Location selection overlay

When choosing the type of guildhall or temple to dedicate, you can search for the relevant profession, religion, or deity by name. For temples, you can also search for the "spheres" associated with the deity or religion, such as "wealth" or "lies".

You can also choose whether to filter out temple or guildhall types that you have already established.

Slab engraving overlay

When choosing a unit to engrave a slab for, you can search for units by name, either in their native language or in English (though only their native name will be displayed). This overlay also adds a filter for showing only units that would need a slab in order to prevent them rising as a ghost.

World overlay

Searching is supported for the Artifacts list when viewing the world map (where you can initiate raids).

5.6.219 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.

5.6.220 spawnunit

Tags: unavailable

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:

```
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!

5.6.221 spectate

Tags: fort | interface

Command: `spectate`

Automatically follow productive dwarves.

Usage

```
enable spectate
spectate
spectate set <setting> <value>
spectate enable|disable <feature>
```

When enabled, the plugin will lock the camera to following the dwarves scurrying around your fort. Every once in a while, it will automatically switch to following a different dwarf, preferring dwarves on z-levels with the highest job activity.

If you have the `auto-disengage` feature disabled, you can switch to a new dwarf immediately by hitting one of the map movement keys (`wasd` by default). To stop following dwarves, bring up *gui/launcher* and run `disable spectate`.

Changes to settings will be saved with your fort, but if *spectate* is enabled when you save the fort, it will disenable itself when you load so you can get your bearings before re-enabling follow mode with `enable spectate` again.

Examples

enable spectate
Starting following dwarves and observing life in your fort.

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 1000
Set the tick interval between camera changes 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, titans, invaders, etc.) (default: disabled)

visiting
Toggle whether to sometimes follow visiting units (eg. diplomats)

Settings

tick-threshold
Set the plugin's tick interval for changing the followed dwarf. (default: 1000)

5.6.222 startdwarf

Tags: embark | fort | armok

Command: `startdwarf`

Change the number of dwarves you embark with.

You must use this tool before you get to the embark preparation screen (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. The value that you set will remain in effect until DF is restarted (or you use *startdwarf* to set a new value).

The maximum number of dwarves you can have is 32,767, but that is far more than the game can handle.

Usage

```
startdwarf <number>
```

Examples

`startdwarf 10`
Start with a few more warm bodies to help you get started.

`startdwarf 1`
Hermit fort! (also see the *hermit* tool for keeping it that way)

`startdwarf 500`
Start with a teeming army of dwarves (leading to immediate food shortage and FPS issues).

Overlay

The vanilla DF screen doesn't provide a way to scroll through the starting dwarves, so if you start with more dwarves than can fit on your screen, this tool provides a scrollbar that you can use to scroll through them. The vanilla list was *not* designed for scrolling, so there is some odd behavior. When you click on a dwarf to set skills, the list will jump so that the dwarf you clicked on will be at the top of the page.

5.6.223 starvingdead

Tags: fort | 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.

Strength lost is proportional to the time until death, all units will have roughly 10% of each of their attributes' values when close to being removed.

In any game, this can be a welcome gameplay feature, but it is especially useful in preventing undead cascades in the caverns in reanimating biomes, where constant combat can lead to hundreds of undead roaming the caverns and destroying your FPS.

Usage

```
enable starvingdead
starvingdead [<options>]
```

Examples

`enable starvingdead`
: Start starving the dead with default settings.

`starvingdead --decay-rate 28`
: Undead will lose strength roughly once a month.

`starvingdead --decay-rate 1 --death-threshold 1`
: Undead will lose strength each day and die after they have spent a month on the map.

Options

`--decay-rate <days>`
Specify how often, in days, undead should lose strength.

`--death-threshold <months>`
How many months should undead lose strength for before being removed.

5.6.224 steam-engine

Tags: unavailable

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.

5.6.225 stockflow

Tags: unavailable

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.

5.6.226 stockpiles

Tags: fort | design | productivity | stockpiles

Command: `stockpiles`

Import, export, or modify stockpile settings.

Commands act upon the stockpile selected in the UI unless another stockpile identifier is specified on the commandline.

You can also specify hauling route stops to import and export "desired items" configuration.

Usage

```
stockpiles [status]
stockpiles list [<search>]
stockpiles import <name> [<options>]
stockpiles export <name> [<options>]
```

Exported settings are saved in the `dfhack-config/stockpiles` folder, where you can view and delete them, if desired. Names can only contain numbers, letters, periods, and underscores.

The names of library settings files are all prefixed by the string `library/`. You can specify library files explicitly by including the prefix, or you can just write the short name to use a player-exported file by that name if it exists, and the library file if it doesn't.

Examples

stockpiles
Shows the list of all your stockpiles and some relevant statistics.

stockpiles list
Shows the list of previously exported stockpile settings files, including the stockpile configuration library.

stockpiles list plants
Shows the list of exported stockpile settings files that include the substring `plants`.

stockpiles import library/plants
Imports the library `plants` settings file into the currently selected stockpile.

`stockpiles import plants`
: Imports a player-exported settings file named `plants`, or the library `plants` settings file if a player-exported file by that name doesn't exist.

`stockpiles import -m enable plants`
: Enables plants in the selected stockpile.

`stockpiles import -m disable cat_food -f tallow`
: Disables all tallow in the selected food stockpile.

`stockpiles export mysettings`
: Export the settings for the currently selected stockpile to a file named `dfhack-config/stockpiles/mysettings.dfstock`.

`stockpiles export mysettings -i categories,types`
: Export the stockpile category and item settings, but ignore the container and general settings. This allows you to import the configuration later without touching the container and general settings of the target stockpile.

`stockpiles export mydumpersettings -r "Stone quantum"`
: Exports the "desired items" for the first stop of the "Stone quantum" hauling route.

`stockpiles export myroutesettings -r "Train porter,2"`
: Exports the "desired items" for the stop with id 2 (the second stop, unless you've previously deleted the original first stop and now this is the first) of the "Train porter" hauling route.

Options

`-s, --stockpile <name or id>`
: Specify a specific stockpile by name or internal ID instead of using the stockpile currently selected in the UI.

`-r, --route <route name or id>[,<stop name or id>]`
: Specify a hauling route and route stop as the target for import/export instead of a stockpile. If not specified, the first route stop is targeted.

`-i, --include <comma separated list of elements to include>`
: When exporting, you can include this option to select only specific elements of the stockpile to record. If not specified, everything is included. When the file is later imported, only the included settings will be modified. The options are explained below in the next section.

`-m, --mode (set|enable|disable)`
: When importing, choose the algorithm used to apply the settings. In `set` mode (the default), the stockpile is cleared and the settings in the file are enabled. In `enable` mode, enabled settings in the file are *added* to the stockpile, but no other settings are changed. In `disable` mode, enabled settings in the file are *removed* from the current stockpile configuration, and nothing else is changed.

`-f, --filter <search>[,<search>...]`
: When importing, only modify the settings that contain at least one of the given substrings.

Configuration elements

The different configuration elements you can include in an exported settings file are:

containers
Max bins, max barrels, and num wheelbarrows.

general
Whether the stockpile takes from links only and whether organic and/or inorganic materials are allowed.

categories
The top-level categories of items that are enabled for the stockpile, like Ammo, Finished goods, or Stone.

types
The elements below the categories, which include the sub-categories, the specific item types, and any toggles the category might have (like Prepared meals for the Food category).

Overlay

This plugin provides a panel that appears when you select a stockpile via an *overlay* widget. You can use it to easily toggle *logistics* plugin features like autotrade, automelt, or autotrain. There are also buttons along the top frame for:

- minimizing the panel (if it is in the way of the vanilla stockpile configuration widgets)
- showing help for the overlay widget in *gui/launcher* (this page)
- configuring advanced settings for the stockpile, such as whether automelt will melt masterworks

The stockpiles settings library

DFHack comes with a library of useful stockpile settings files that are ready for import. If the stockpile configuration that you need isn't directly represented, you can often use the `enable` and `disable` modes and/or the `filter` option to transform an existing saved stockpile setting. Some stockpile configurations can only be achieved with filters since the stockpile lists are different for each world. For example, to disable all tallow in your main food stockpile, you'd run this command:

```
stockpiles import cat_food -m disable -f tallow
```

Top-level categories

Each stockpile category has a file that allows you to enable or disable the entire category, or with a filter, any matchable subset thereof:

```
cat_ammo
cat_animals
cat_armor
cat_bars_blocks
cat_cloth
cat_coins
cat_corpses
cat_finished_goods
cat_food
```

(continues on next page)

(continued from previous page)

```
cat_furniture
cat_gems
cat_leather
cat_refuse
cat_sheets
cat_stone
cat_weapons
cat_wood
```

In addition, there are files for `all`, which includes all categories except refuse and corpses (mirroring the "all" configuration in-game), and `everything`, which really includes all categories.

For many of the categories, there are also flags, subcategory prefixes, and item properties that you can match with filters. In addition, there are normally at least a few convenient pre-made settings files that manipulate interesting category subsets.

Cross-category stockpile adjustments

Settings files:

```
artifacts
masterworks
```

Example command for a meltable weapons stockpile:

```
stockpiles import cat_weapons
stockpiles import -m disable cat_weapons -f other/
stockpiles import -m disable artifacts
stockpiles import -m disable masterworks
```

Ammo stockpile adjustments

Subcategory prefixes:

```
type/
mats/
other/
core/
total/
```

Settings files:

```
bolts
metalammo
boneammo
woodammo
```

Example commands for a stockpile of metal bolts:

```
stockpiles import cat_ammo -f mats/,core/,total/
stockpiles import -m enable bolts
```

Animal stockpile adjustments

Flags:

```
cages
traps
```

Properties:

```
tameable
```

Settings files:

```
cages
traps
```

Example commands for a stockpile of empty cages:

```
stockpiles import cages
```

Or, using the flag for the same effect:

```
stockpiles import cat_animals -f cages
```

Armor stockpile adjustments

Flags and subcategory prefixes:

```
nouse
canuse
body/
head/
feet/
hands/
legs/
shield/
mats/
other/
core/
total/
```

Settings files:

```
metalarmor
otherarmor
ironarmor
bronzearmor
copperarmor
steelarmor
usablearmor
unusablearmor
```

Example commands for a stockpile of sub-masterwork meltable armor:

```
stockpiles import cat_armor
stockpiles import -m disable -f other/,core/mas,core/art cat_armor
```

Bar stockpile adjustments

Subcategory prefixes:

```
mats/bars/
other/bars/
mats/blocks/
other/blocks/
```

Settings files:

```
bars
metalbars
ironbars
pigironbars
steelbars
otherbars
coal
potash
ash
pearlash
soap
blocks
```

Example commands for a stockpile of blocks:

```
stockpiles import blocks
```

Cloth stockpile adjustments

Subcategory prefixes:

```
thread/silk/
thread/plant/
thread/yarn/
thread/metal/
cloth/silk/
cloth/plant/
cloth/yarn/
cloth/metal/
```

Settings files:

```
thread
adamantinethread
cloth
adamantinecloth
```

Notes:

- **`thread` and `cloth` settings files set all materials that are not** adamantine.

Corpse stockpile adjustments

Properties:

```
tameable
```

Finished goods stockpile adjustments

Subcategory prefixes:

```
type/
mats/
other/
core/
total/
```

Settings files:

```
stonetools
woodtools
crafts
goblets
toys
```

Example commands for a toy stockpile:

```
stockpiles import cat_finished_goods -f mats/,other/,core/,total/
stockpiles import -m enable toys
```

Food stockpile adjustments

Flags and subcategory prefixes:

```
preparedmeals
meat/
fish/prepared/
fish/unprepared/
egg/
plants/
drink/plant/
drink/animal/
cheese/plant/
cheese/animal/
seeds/
leaves/
powder/plant/
powder/animal/
```

(continues on next page)

(continued from previous page)

```
glob/
liquid/plant/
liquid/animal/
liquid/misc/
paste/
pressed/
```

Settings files:

```
preparedmeals
unpreparedfish
plants
booze
seeds
dye
miscliquid
wax
```

Example commands for a kitchen ingredients stockpile:

```
stockpiles import cat_food -f meat/,fish/prepared/,egg/,cheese/,leaves/,powder/,glob/,
↪liquid/plant/,paste/,pressed/
stockpiles import cat_food -m enable -f milk,royal_jelly
stockpiles import dye -m disable
stockpiles import cat_food -m disable -f tallow,thread,liquid/misc/
```

Furniture stockpile adjustments

Subcategory prefixes:

```
type/
mats/
other/
core/
total/
```

Settings files:

```
pots
barrels
bags
buckets
sand
```

- Because of the limitations of Dwarf Fortress, `bags` cannot distinguish between empty bags and bags filled with gypsum powder.

Example commands for a sand bag stockpile:

```
stockpiles import cat_furniture
stockpiles import cat_furniture -m disable -f type/
stockpiles import sand -m enable
```

Gem stockpile adjustments

Subcategory prefixes:

```
mats/rough/
mats/cut/
other/rough/
other/cut/
```

Settings files:

```
roughgems
roughglass
cutgems
cutglass
cutstone
```

Refuse stockpile adjustments

Flags and subcategory prefixes:

```
rawhide/fresh
rawhide/rotten
type/
corpses/
bodyparts/
skulls/
bones/
hair/
shells/
teeth/
horns/
```

Properties:

```
tameable
```

Settings files:

```
rawhides
tannedhides
usablehair
```

Notes:

- `usablehair` Only hair and wool that can make usable clothing is included, i.e. from sheep, llamas, alpacas, and trolls.

Example commands for a craftable refuse stockpile:

```
stockpiles import cat_refuse -f skulls/,bones/,shells',teeth/,horns/
stockpiles import usablehair -m enable
```

Sheet stockpile adjustments

Subcategory prefixes:

```
paper/
parchment/
```

Stone stockpile adjustments

Settings files:

```
metalore
ironore
economic
flux
plasterproducing
coalproducing
otherstone
bauxite
clay
```

Weapon stockpile adjustments

Flags and subcategory prefixes:

```
nouse
canuse
type/weapon/
type/trapcomp/
mats/
other/
core/
total/
```

Settings files:

```
metalweapons
stoneweapons
otherweapons
trapcomponents
ironweapons
silverweapons
bronzeweapons
copperweapons
steelweapons
platinumweapons
adamantineweapons
usableweapons
unusableweapons
```

Example commands for a non-metallic trap components stockpile:

```
stockpiles import cat_weapons
stockpiles import cat_weapons -m disable -f type/weapon/
stockpiles metalweapons -m disable
```

5.6.227 stocks

Tags: unavailable

Command: `stocks`

Enhanced fortress stock management interface.

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.

5.6.228 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.

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.

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.
```

Fig. 1: The above-ground part of the fortress *Roadtruss*.

```
A complete listing of valid actions and key values can be found at
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

5.6.229 strangemood

Tags: fort | armok | units

Command: `strangemood`

Trigger a strange mood.

Usage

```
strangemood [<options>]
```

Example

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", "leatherworker", "bonecarver", "bowyer", and "mechanic".

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

5.6.230 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, though, is to strip the weapons and armor from caged prisoners. After you run `stripcaged`, 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
stripcaged items|weapons|armor|all [here|<cage id> ...] [<options>]
```

Examples

`stripcaged list`
Display a list of all cages and their item contents.

`stripcaged all`
Dump all items in all cages, 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 --skip-forbidden`
Dumps unforbidden armor equipped by the caged creature in the selected cage.

`stripcaged all 25321 34228`
Dumps all items out of the specified cages.

`stripcaged items here --include-pets --include-vermin`
Dumps loose items in the selected cage, including any tamed/untamed vermin.

Options

--include-pets, --include-vermin
Live tame (pets) and untamed vermin are considered items by the game. They are normally excluded from dumping since that risks them escaping or dying from your cats. Use these options to dump them anyway.

-f, --skip-forbidden
Items to be marked for dumping are unforbidden by default. Use this option to instead only act on unforbidden items, and leave forbidden items forbidden. This allows you to, for example, manually unforbid high-value items from the stocks menu (like steel) and then have `stripcaged` just act on the unforbidden items.

5.6.231 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.

Sets each action timer every tick to 1, effectively giving all actions such as work, movement, combat, etc. no cooldown.

Usage

superdwarf add
Give superspeed to selected creature.

superdwarf add <ID>
Give superspeed to a creature by its unit ID.

superdwarf all
Give superspeed to all of your citizens.

superdwarf del
Remove superspeed from selected creature.

superdwarf del <ID>
Remove superspeed to a creature by its unit ID.

superdwarf clear
Remove superspeed from all creatures.

superdwarf list
List creatures with superspeed.

5.6.232 suspend

Tags: fort | productivity | jobs

Command: `suspend`

Suspends building construction jobs.

This tool will suspend jobs. It can either suspend all the current jobs, or only construction jobs that are likely to block other jobs. When building walls, it's common that wall corners get stuck because dwarves build the two adjacent walls before the corner. The `--onlyblocking` option will only suspend jobs that can potentially lead to this situation.

See *suspendmanager* in *gui/control-panel* to automatically suspend and unsuspend jobs.

Usage

```
suspend
```

Options

`-b`, `--onlyblocking`
Only suspend jobs that are likely to block other jobs.

Note: `--onlyblocking` does not check pathing (which would be very expensive); it only looks at immediate neighbours. As such, it is possible that this tool will miss suspending some jobs that prevent access to other farther away jobs, for example when building a large rectangle of solid walls.

5.6.233 suspendmanager

Tags: fort | auto | jobs

Command: `suspendmanager`

Intelligently suspend and unsuspend jobs.

Command: `unsuspend`

Resume suspended building construction jobs.

When enabled, `suspendmanager` will watch your active jobs and:

- unsuspend jobs that have become suspended due to inaccessible materials, items temporarily in the way, or worker dwarves getting scared by wildlife

- suspend most construction jobs that would prevent a dwarf from reaching another construction job, such as when building a wall corner or high walls
- suspend construction jobs on top of smoothing, engraving, or track carving designations. This prevents the construction job from being completed first, which would cause the designation to be lost.
- suspend construction jobs that would cave in immediately on completion, such as when building walls or floors next to grates/bars.

Usage

enable suspendmanager
Start monitoring jobs.

suspendmanager
Display the current status

suspendmanager set preventblocking (true|false)
Prevent construction jobs from blocking each others (enabled by default). See *suspend*.

unsuspend [-s|--skipblocking] [-q|--quiet]
Perform a single cycle, suspending and unsuspending jobs as described above, regardless of whether *suspendmanager* is enabled.

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.

Options

-q, --quiet
Suppress summary output.

-s, --skipblocking
Also resume jobs that may block other jobs.

Overlay

This plugin also provides an overlay that is managed by the *overlay* framework. When the overlay is enabled, an icon or letter will appear over suspended buildings:

- A clock icon (green `P` in ASCII mode) indicates that the building is still in *planning mode* and is waiting on materials. The building will be unsuspended for you when those materials become available.
- A white `x` means that the building has been suspended by *suspendmanager*. If you select the building in the UI, you will see the reason for the suspension.
- A yellow `x` means that the building is suspended and no DFHack tool is managing the suspension. You can unsuspend it manually or with the *unsuspend* command.
- A red `X` means that the building has been re-suspended multiple times. You might need to look into whatever is preventing the building from being built (e.g. the building material for the building is inaccessible or there is an in-use item blocking the building site).

Note that in ASCII mode the letter will only appear when the game is paused since it takes up the whole tile and hides the underlying building. In graphics mode, the icon only covers part of the building and so can always be visible.

5.6.234 sync-windmills

Tags: fort | buildings

Command: `sync-windmills`

Synchronize or randomize windmill movement.

Windmills cycle between two graphical states to simulate movement. This is the polarity of the appearance. Each windmill also has a timer that controls when the windmill switches polarity. Each windmill's timer starts from zero at the instant that it is built, so two different windmills will rarely have exactly the same state. This tool can adjust the alignment of polarity and timers across your active windmills to your preference.

Note that this tool will not affect windmills that have just been activated and are still rotating to adjust to the regional wind direction.

Usage

```
sync-windmills [<options>]
```

Examples

sync-windmills
Synchronize movement of all active windmills.

sync-windmills -r
Randomize the movement of all active windmills.

Options

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

-r, --randomize
Randomize the polarity and timer value for all windmills.

-t, --timing-only
Randomize windmill polarity, but synchronize windmill timers.

5.6.235 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.

5.6.236 tailor

Tags: fort | auto | workorders

Command: `tailor`

Automatically keep your dwarves in fresh clothing.

Once a day, this plugin will scan the clothing situation in 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.

If you are wondering whether you should enable *autoclothing* instead, see the head-to-head comparsion in the *autoclothing* docs.

Usage

```
enable tailor
tailor [status]
tailor now
tailor materials <material> [<material> ...]
tailor confiscate [true|false]
```

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. By default, `tailor` will "confiscate" (i.e., remove ownership and mark for dumping) equipped tattered clothing once replacements are available. This can be changed using `tailor confiscate`. The default behavior minimizes the time you dwarves keep wearing worn-out clothing, minimizing the number of negative thoughts incurred by this. However, it will also result in tattered clothing being brought to your "garbage" zones and forbidden, which may be undesirable.

`tailor` supports adamantine cloth (using the `materials` keyword `adamantine`) but does not use it by default, as most players find adamantine too precious to routinely make into cloth. `tailor` does not support modded "cloth" types which utilize custom reactions for making clothing out of those cloth types.

Examples

`enable tailor`
: Start replacing tattered clothes with default settings.

`tailor now`
: Run a scan and order cycle right now, regardless of whether the plugin is enabled.

`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.

Caveats

Modded cloth-like materials are not supported because custom reactions do not support being sized for non-dwarf races. The game only supports sizing the built-in default make-clothing or make-armor reactions.

5.6.237 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

5.6.238 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
```

5.6.239 tidlers

Tags: unavailable

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
```

5.6.240 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.

5.6.241 timestream

Tags: unavailable

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.

5.6.242 title-folder

Tags: unavailable

Displays the DF folder name in the window title bar.

Usage

```
enable title-folder
```

5.6.243 toggle-kbd-cursor

Tags: interface

Command: `toggle-kbd-cursor`

Toggles the keyboard cursor.

Keybinding: AltK in `dwarfmode`

This tool simply toggles the keyboard cursor so you can quickly switch it on when you need it. Many other tools, like *autodump*, need a keyboard cursor for selecting a target tile. Note that you'll still need to enter an interface mode where the keyboard cursor is visible, like mining mode or dumping mode, in order to use the cursor.

Usage

```
toggle-kbd-cursor
```

5.6.244 trackstop

Tags: fort | buildings | interface

Command: `trackstop`

Add dynamic configuration options for track stops.

This script provides 2 overlays that are managed by the *overlay* framework. The script does nothing when executed. The trackstop overlay allows the player to change the friction and dump direction of a selected track stop after it has been constructed. The rollers overlay allows the player to change the roller direction and speed of a selected roller after it has been constructed.

5.6.245 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
```

5.6.246 tubefill

Tags: fort | armok | map

Command: `tubefill`

Replenishes mined-out adamantine.

This tool replaces mined-out tiles of adamantine spires with fresh, undug adamantine walls, ready to be re-harvested. Empty tiles within the spire that used to contain special gemstones, obsidian, water, or magma will also be replaced with fresh adamantine. Adamantine spires that were originally hollow will be left hollow. See below for more details.

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!

5.6.247 twaterlvl

Tags: interface

Command: `twaterlvl`

Show/hide numeric liquid depth on the map.

You can use this tool to toggle between displaying/not displaying liquid depth as numbers on the map.

Usage

```
twaterlvl
```

5.6.248 tweak

Tags: fort | bugfix | fps | gameplay | interface

Command: `tweak`

A collection of tweaks and bugfixes.

Usage

```
tweak [list]
tweak <command> [disable] [quiet]
```

Run the `tweak` command with the tweak name to enable its effects. Add a `disable` keyword to disable them, and a `quiet` keyword to prevent printing of non-error output.

Examples

`tweak eggs-fertile`
Enable the `eggs-fertile` tweak.

`tweak fast-heat quiet`
Enable the `fast-heat` tweak, but don't print anything to the console to say that it has been enabled.

`tweak fast-heat disable quiet`
Disable the `fast-heat` tweak, and be quiet about it.

Commands

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

`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 in-game years.

`eggs-fertile`
Displays an indicator on fertile eggs.

`fast-heat`
Improves temperature update performance by ensuring that 1 degree of item temperature is crossed in no more than 100 ticks when updating from the environment temperature. This reduces the time it takes for temperature to reach equilibrium and improves FPS when there are many items.

`flask-contents`
: Names filled waterskins, flasks, and vials according to their contents, the same way other containers such as barrels, bins, and cages are named. (Bug 4914)

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

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

5.6.249 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>
```

5.6.250 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
```

5.6.251 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

-u, --include-unreachable
Allows the tool to unforbid unreachable items.

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

-X, --include-worn
Include worn (X) and tattered (XX) items when unforbidding.

5.6.252 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.

5.6.253 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. If multiple units claim the same item, the item will be unassigned from all units that are not already wearing the item. If this happens, you'll have to click the "Update equipment" button on the Squads "Equip" screen in order for them to get new equipment assigned.

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.

When generating a report of conflicts, items that simply haven't been picked up yet or uniform components that haven't been assigned by DF are not considered conflicts and are not included in the report.

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 items from the uniform assignment if someone else has a claim on them. This will also remove items from containers and place them on the ground, ready to be claimed.

--multi
Attempt to fix issues with uniforms that allow multiple items per body part.

Overlay

This script adds a small link to the squad equipment page that will run `uniform-unstick --all` and show the report when clicked. After reviewing the report, you can right click to exit and do nothing or you can click the "Try to resolve conflicts" button, which runs the equivalent of `uniform-unstick --all --drop --free`. If any items are unassigned (they'll turn red on the equipment screen), hit the "Update Equipment" button to reassign equipment.

5.6.254 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.

5.6.255 unretire-anyone

Tags: adventure | embark | armok

Command: `unretire-anyone`

Adventure as any living historical figure.

Keybinding: CtrlA in `setupadventure`

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 list of people to choose from during character creation (the "Specific person" option on the race list). 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.

5.6.256 view-item-info

Tags: unavailable

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.

5.6.257 view-unit-reports

Tags: unavailable

Command: `view-unit-reports`

Show combat reports for a unit.

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
```

5.6.258 warn-stranded

Tags: fort | units

Command: `warn-stranded`

Reports citizens who can't reach any other citizens.

If any groups of sane fort citizens are stranded from the main (largest) group, you'll get a warning dialog telling you which citizens are isolated. This gives you a chance to rescue them before they get overly stressed or starve.

There is a command line interface that can print status of citizens without pausing or bringing up a window.

If there are citizens that you are ok with stranding (say, you have isolated a potential werebeast or vampire), you can mark them as ignored so they won't trigger a warning.

This tool is integrated with *gui/notify* to automatically show notifications when a stranded unit is detected.

Usage

```
warn-stranded
warn-stranded status
warn-stranded clear
warn-stranded (ignore|unignore) <unit id>
warn-stranded (ignoregroup|unignoregroup) <group id>
```

Examples

warn-stranded
Standard command that checks citizens and pops up a warning if any are stranded. Does nothing when there are no unignored stranded citizens.

warn-stranded status
List all groups of stranded citizens and all ignored citizens. Also shows individual unit ids.

warn-stranded clear
Clear (unignore) all ignored citizens.

warn-stranded ignore 15343
Ignore citizen with unit id 15343.

warn-stranded ignoregroup 2
Ignore stranded citizen group 2.

5.6.259 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.

5.6.260 work-now

Tags: fort | gameplay | jobs

Command: `work-now`

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 them to pick up a new task quicker.

Usage

```
enable work-now
work-now [status]
```

5.6.261 workflow

Tags: unavailable

Command: `workflow`

Manage automated item production rules.

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.

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).

5.6.262 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\":\"MilkCreature\",\"item_conditions\":[{\"condition\":\"AtLeast\",\"value\":5,\"flags\":[\"empty\"],\"item_type\":\"BUCKET\"}]}"]
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. The first matching existing order will be modified to have the desired quantity remaining. If the desired quantity is negative, the existing order will be removed. An empty `amount_total` implies `"__reduce_amount": true`.

5.6.263 xlsxreader

Tags: dev

Provides a Lua API for reading xlsx files.

See *xlsxreader* for details.

5.6.264 zone

Tags: unavailable

Command: `zone`

Manage activity zones, cages, and the animals therein.

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.

Overlays

Animal Assignment

Advanced unit selection is available via an *overlay* widget that appears when you select a cage, restraint, pasture zone, or pit/pond zone.

In the window that pops up when you click the hotkey hint or hit the hotkey on your keyboard, you can:

- search for units by name
- **sort or filter by status (Assigned here, Pastured elsewhere, On restraint, On** display in cage, In movable cage, or Roaming)
- **sort or filter by disposition (Pet, Domesticated, Partially trained, Wild** (trainable), Wild (untrainable), or Hostile)
- sort by gender
- sort by name
- filter by whether the unit lays eggs
- filter by whether the unit needs a grazing area

The window is fully navigatable via keyboard or mouse. Hit Enter or click on a unit to assign/unassign it to the currently selected zone or building. Shift click to assign/unassign a range of units.

You can also keep the window open and click around on different cages, restraints, pastures, or pit/ponds, so you can manage multiple buildings/zones without having to close and reopen the window.

Just like all other overlays, you can disable this one in *gui/control-panel* on the Overlays tab if you don't want the option of using it.

Location retirement

When viewing location (temple, guildhall, tavern, etc.) details, there is now a "Retire location" button that you can use to remove the location from the list of available locations for your fort.

Before you can retire the location, you have to remove all units assigned occupations for that location and you have to detach the location from any zones. Once you confirm retirement, the location will be removed from the list.

5.6.265 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
```

5.6.266 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.

5.6.267 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
```

5.6.268 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
```

5.6.269 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
```

5.6.270 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.

5.6.271 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
```

5.6.272 devel/cmptiles

Tags: unavailable

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]
```

5.6.273 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.

5.6.274 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>
```

5.6.275 devel/dump-tooltip-ids

Tags: dev

Command: `devel/dump-tooltip-ids`

Generate main_hover_instruction enum XML structures.

This script generates the contents of the `main_hover_instruction` enum and attrs, then cross-checks with the currently-built enum attrs to determine the correct enum item names. This is intended to catch cases where items move around.

For example, if DFHack has:

```
<enum-item name='ArenaMud'>
    <item-attr name='caption' value='Add mud to the arena.'/>
</enum-item>
```

as item 500, but we detect that caption at position 501 in `main_interface.hover_instruction`, then the output produced by the script will include the above element at position 501 instead of 500.

Before running this script, the size of `main_interface.hover_instruction` must be aligned properly with the loaded verison of DF so the array of strings can be read.

Usage

```
devel/dump-tooltip-ids
```

5.6.276 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

5.6.277 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
```

5.6.278 devel/find-offsets

Tags: unavailable

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

5.6.279 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.

5.6.280 devel/find-twbt

Tags: unavailable

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
```

5.6.281 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
```

5.6.282 devel/inject-raws

Tags: unavailable

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
```

5.6.283 devel/input-monitor

Tags: dev

Command: `devel/input-monitor`

Live monitor and logger for input events.

This UI allows you to discover how DF is interpreting input from your keyboard and mouse device.

The labels for Shift, Ctrl, and Alt light up when those modifier keys are being held down.

Similar lables for left, middle, and right mouse buttons light up when any of those buttons are being held down.

The input stream panel shows the keybindings that are being triggered. You can resize the window to see more of the stream history. The events are also logged to the *external console*, if you need a more permanent record of the stream.

Since right click is intercepted, it cannot be used to close this window. Instead, hit `Esc` twice in a row or click twice on the exit button.

Usage

```
devel/input-monitor
```

5.6.284 devel/inspect-screen

Tags: dev

Command: `devel/inspect-screen`

Show glyph, color, and texture info for screen and map tiles.

This script pops up a panel that shows all known information about the tile under the mouse cursor. This includes the displayed character or text for the selected tile and foreground/background color information. All textures associated with the tile and flags associated with those textures are also displayed.

You can toggle between seeing information for the UI grid tiles or the map tiles. Click on a tile to freeze the displayed information so you can scroll the information about the tile or move/resize the inspector window without changing the active tile.

Usage

```
devel/inspect-screen
```

5.6.285 devel/kill-hf

Tags: unavailable

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.

5.6.286 devel/light

Tags: unavailable

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.

5.6.287 devel/list-filters

Tags: unavailable

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
```

5.6.288 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 [<address> ...] [<name|pattern> ...]
```

Examples

devel/lsmem 0x1234 5678 90ab
: List any ranges containing the addresses `0x1234`, `0x5678`, or `0x90ab`. Addresses are interpreted as hex; the `0x` prefix is optional.

devel/lsmem dwarf g_src
: List any ranges corresponding to files matching `dwarf` or `g_src` (case-insensitive).

devel/lsmem .+
: List any ranges with non-empty filenames. Any Lua patterns are allowed.

5.6.289 devel/lua-example

Tags: unavailable

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.

5.6.290 devel/luacov

Tags: unavailable

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.

5.6.291 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
```

5.6.292 devel/nuke-items

Tags: unavailable

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
```

5.6.293 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
```

5.6.294 devel/prepare-save

Tags: unavailable

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
```

5.6.295 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>]
```

5.6.296 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>]
```

5.6.297 devel/print-event

Tags: unavailable

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.

5.6.298 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.

5.6.299 devel/save-version

Tags: dev

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
```

5.6.300 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.

5.6.301 devel/scan-vtables

Tags: dev

Command: `devel/scan-vtables`

Scan for and print likely vtable 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 scans for likely vtables in memory pages mapped to the DF executable, and prints them in a format ready for inclusion in `symbols.xml`

Usage

```
devel/scan-vtables
```

5.6.302 devel/scanitemother

Tags: dev

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
```

5.6.303 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.

5.6.304 devel/spawn-unit-helper

Tags: dev

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`

5.6.305 devel/test-perlin

Tags: unavailable

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.

5.6.306 devel/tile-browser

Tags: dev

Command: `devel/tile-browser`

Browse graphical tile textures by their texpos values.

This script pops up a panel that shows a page of 1000 textures. You can change the starting texpos index with `CtrlA` or scan forward or backwards in increments of 1000 with `Shift Up/Down`.

Textures are resized to the dimensions of the interface grid (8px by 12px), so map textures (nominally 32px by 32px) will be squished. If no texture is assigned to the texpos index, the window will be transparent at that spot.

Usage

```
devel/tile-browser
```

5.6.307 devel/unit-path

Tags: unavailable

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
```

5.6.308 devel/visualize-structure

Tags: dev

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
```

5.6.309 devel/watch-minecarts

Tags: unavailable

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
```

5.6.310 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
```

5.6.311 fix/civil-war

Tags: fort | bugfix

Command: `fix/civil-war`

Removes a civil war.

This script removes a negative relation with your own government (civil war).

If caravans/migrants do not return after fixing the civil war, try running "force Caravan" and trading to increase wealth exported to parent civ.

Usage

```
fix/civil-war
```

5.6.312 fix/corrupt-equipment

Tags: unavailable

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

5.6.313 fix/corrupt-jobs

Tags: fort | bugfix

Command: `fix/corrupt-jobs`

Removes jobs with an id of -1 from units.

This fix cleans up corrupt jobs so they don't cause crashes. It runs automatically on fort load, so you don't have to run it manually.

Usage

```
fix/corrupt-jobs
```

5.6.314 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.

It also supports scanning burrows and cleaning out dead units from burrow assignments. The vanilla UI doesn't provide any way to remove dead units, and the dead units artificially increase the reported count of units that are assigned to the burrow.

Usage

```
fix/dead-units [--active] [-q]
fix/dead-units --burrow [-q]
```

Options

`--active`
: Scrub units that have been dead for more than a month from the `active` vector. This is the default if no option is specified.

`--burrow`
: Scrub dead units from burrow membership lists.

`-q`, `--quiet`
: Surpress console output (final status update is still printed if at least one item was affected).

5.6.315 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.

5.6.316 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.

This tool also fixes over-full buckets that are blocking well operations.

Usage

```
fix/dry-buckets
```

5.6.317 fix/empty-wheelbarrows

Tags: fort | bugfix | items

Command: `fix/empty-wheelbarrows`

Empties stuck items from wheelbarrows.

Empties all wheelbarrows which contain rocks that have become 'stuck' in them.

This works around the issue encountered with Bug 6074, and should be run if you notice wheelbarrows lying around with rocks in them that aren't being used in a task. This script is set to run periodically by default in *gui/control-panel*.

Usage

```
fix/empty-wheelbarrows [<options>]
```

Examples

fix/empty-wheelbarrows
Empties all items, listing all wheelbarrows emptied and their contents.

fix/empty-wheelbarrows --dry-run
Lists all wheelbarrows that would be emptied and their contents without performing the action.

fix/empty-wheelbarrows --quiet
Does the action while surpressing output to console.

Options

-q, --quiet
Surpress console output (final status update is still printed if at least one item was affected).

-d, --dry-run
Dry run, don't commit changes.

5.6.318 fix/engravings

Tags: fort | bugfix

Command: `fix/engravings`

Fixes unengravable corrupted tiles so they are able to be engraved.

When constructing a new wall or new floor over a previously engraved tile, the tile may become corrupted and unengravable. This fix detects the problem and resets the state of those tiles so they may be engraved again.

Usage

```
fix/engravings [<options>]
```

Options

-q, --quiet
Only output status when something was actually fixed.

5.6.319 fix/general-strike

Tags: fort | bugfix

Command: `fix/general-strike`

Prevent dwarves from getting stuck and refusing to work.

This script attempts to fix known causes of the "general strike bug", where dwarves just stop accepting work and stand around with "No job".

This script is set to run periodically by default in *gui/control-panel*.

Usage

```
fix/general-strike [<options>]
```

Options

-q, --quiet
Only output status when something was actually fixed.

5.6.320 fix/item-occupancy

Tags: unavailable

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

5.6.321 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
```

5.6.322 fix/noexert-exhaustion

Tags: fort | bugfix | units

Command: `fix/noexert-exhaustion`

Prevents NOEXERT units from getting tired when training.

This tool zeroes the "exhaustion" counter of all NOEXERT units (e.g. Vampires, Necromancers, and Intelligent Undead), fixing any that are stuck 'Tired' from an activity that doesn't respect NOEXERT. This is not a permanent fix - the issue will reoccur next time they partake in an activity that does not respect the NOEXERT tag.

Running this regularly works around Bug 8389, which permanently debuffs NOEXERT units and prevents them from properly partaking in military training. It should be run if you notice Vampires, Necromancers, or Intelligent Undead becoming 'Tired'. Enabling this script via *control-panel* or *gui/control-panel* will run it often enough to prevent NOEXERT units from becoming 'Tired'.

Usage

```
fix/noexert-exhaustion
```

Technical details

Units with the NOEXERT tag ignore most sources of physical exertion, and have no means to recover from it. Individual Combat Drill seems to add approximately 50 'Exhaustion' every 9 ticks, ignoring NOEXERT. Units become Tired at 2000 Exhaustion, and switch from Individual Combat Drill to Individual Combat Drill/Resting at 3000. Setting the Exhaustion counter of every NOEXERT-tagged unit to 0 every 350 ticks should prevent them from becoming Tired from Individual Combat Drill.

5.6.323 fix/ownership

Tags: fort | bugfix | units

Command: `fix/ownership`

Fixes instances of units claiming the same item or an item they don't own.

Due to a bug a unit can believe they own an item when they actually do not.

When enabled in *gui/control-panel*, *fix/ownership* will run once a day to check citizens and residents and make sure they don't mistakenly own an item they shouldn't.

This should help issues of units getting stuck in a "Store owned item" job.

Usage

```
fix/ownership
```

5.6.324 fix/population-cap

Tags: unavailable

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.

5.6.325 fix/protect-nicks

Tags: fort | bugfix | units

Command: `fix/protect-nicks`

Fix nicknames being erased or not displayed.

Due to a bug, units nicknames are not displayed everywhere and are occasionally lost.

fix/protect-nicks will save the nicknames of the units once an in-game day. It works by setting the same nickname to the unit's corresponding "historical figure", which was the behavior on pre-Steam releases.

After running it, the nicknames are properly displayed in locations such as legends or image descriptions. Additionally, they are no longer lost after killing a forgotten beast or retiring a fort.

Usage

enable fix/protect-nicks
Enable daily saving of the nicknames.

disable fix/protect-nicks
Disable the daily check.

fix/protect-nicks now
Immediately save the nicknames.

5.6.326 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* to dismiss the caravan followed by `force Caravan` to create a new one may work better.

Usage

```
fix/retrieve-units
```

5.6.327 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
```

5.6.328 fix/stuck-instruments

Tags: fort | bugfix | items

Command: `fix/stuck-instruments`

Allow bugged instruments to be interacted with again.

Fixes instruments that were picked up for a performance, but were instead simulated and are now stuck permanently in a job that no longer exists.

This works around the issue encountered with Bug 9485, and should be run if you notice any instruments lying on the ground that seem to be stuck in a job.

Usage

fix/stuck-instruments
Fixes item data for all stuck instruments on the map.

fix/stuck-instruments -n, fix/stuck-instruments --dry-run
List how many instruments would be fixed without performing the action.

5.6.329 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.

5.6.330 fix/stuck-worship

Tags: fort | bugfix | units

Command: `fix/stuck-worship`

Prevent dwarves from getting stuck in Worship! states.

Dwarves that need to pray to multiple deities sometimes get stuck in a state where they repeatedly fulfill the need to pray/worship one deity but ignore the others. The intense need to pray to the other deities causes the dwarf to start a purple (uninterruptible) `Worship!` activity, but since those needs are never satisfied, the dwarf becomes stuck and effectively useless. More info on this problem at Bug 10918.

This fix analyzes all units that are actively praying/worshipping and detects when they have become stuck (or are on the path to becoming stuck). It then adjusts the distribution of need so when the dwarf finishes their current prayer, a different prayer need will be fulfilled.

This fix will run automatically if it is enabled in the `Bugfixes` tab in *gui/control-panel*. If it is disabled there, you can still run it as needed. You'll know it worked if your units go from `Worship!` to a prayer to some specific deity or if the unit just stops praying altogether and picks up another task.

Usage

```
fix/stuck-worship
```

5.6.331 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
```

5.6.332 fix/tile-occupancy

Tags: unavailable

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
```

5.6.333 gui/advfort

Tags: unavailable

Command: `gui/advfort`

Perform fort-like jobs in adventure mode.

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:

5.6.334 gui/aquifer

Tags: fort | armok | map

Command: `gui/aquifer`

View, add, remove, or modify aquifers.

This is the interactive GUI for the *aquifer* tool. While *gui/aquifer* is open, aquifer tiles will be highlighted as per the *dig.warmdamp* overlay (but unlike that overlay, only aquifer tiles are highlighted, not "just damp" tiles or warm tiles). Note that "just damp" tiles will still be highlighted if they are otherwise already visible.

You can draw boxes around areas of tiles to alter their aquifer properties, or you can use the `CtrlA`` shortcut to affect entire layers at a time.

If you want to see where the aquifer tiles are so you can designate digging, please run *gui/reveal*. If you only want to see the aquifer tiles and not reveal the caverns or other tiles, please run *gui/reveal –aquifers-only* instead.

Usage

```
gui/aquifer
```

5.6.335 gui/autobutcher

Tags: fort | auto | fps | animals

Command: `gui/autobutcher`

Automatically butcher excess livestock.

This is an in-game interface for *autobutcher*, which allows you to set thresholds for how many animals you want to keep and the extras will be automatically butchered.

Usage

```
gui/autobutcher
```

5.6.336 gui/autochop

Tags: fort | auto | plants

Command: `gui/autochop`

Auto-harvest trees when low on stockpiled logs.

This is the configuration interface for the *autochop* plugin. You can configure which burrows will be used for tree chopping, your target stock quantities, and other options. You can also see how many logs you have, the number of logs that the trees on the map are estimated to yield, and how many of each are currently inaccessible.

Usage

```
gui/autochop
```

5.6.337 gui/autodump

Tags: fort | armok | items

Command: `gui/autodump`

Teleport or destroy items.

Keybinding: CtrlH in `dwarfmode`

This is a general point and click interface for teleporting or destroying items. By default, it will teleport items you have marked for dumping, but if you draw boxes around items on the map, it will act on the selected items instead. Double-click anywhere on the map to teleport the items there. Be wary (or excited) that if you teleport the items into an unsupported position (e.g. mid-air), then they will become projectiles and fall.

There are options to include or exclude forbidden items, items that are currently tagged as being used by an active job, and items dropped by traders. If trader items are included, the `trader` flag will be cleared upon teleport so the items can be used.

Usage

```
gui/autodump
```

Destroying items

This tool also allows you to destroy the target items instead of teleporting them. When you click the destroy button (or hit the hotkey), *gui/autodump* will force-pause the game and enable an "Undo" button, just in case you want those items back. Once you exit the *gui/autodump* tool, those items will be unrecoverable.

5.6.338 gui/autofish

Tags: fort | auto | labors

Command: `gui/autofish`

Auto-manage fishing labors to control your stock of fish.

This is the configuration interface for the *autofish* module. You can configure the number of fish you'd like to keep in stock, and whether or not the script should also count your raw fish. You can also check whether or not autofish is currently fishing or not.

Usage

```
gui/autofish
```

5.6.339 gui/autogems

Tags: unavailable

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
```

5.6.340 gui/biomes

Tags: fort | inspection | map

Command: `gui/biomes`

Visualize and inspect biome regions on the map.

This script shows the boundaries of the biome regions on the map. Hover over a biome entry in the list to get detailed info about it.

Note that up in mid-air, there may be additional biomes inherited from neighboring embark squares due to DF Bug 8781. This does not usually affect the player unless:

- You build up into the sky, cast obsidian to make natural flooring, muddy it, and designate a farm plot
- The inherited sky biome is evil and has an effect on fliers that happen to enter its space (e.g. avian wildlife can unexpectedly get zombified or drop dead from syndromes)

Usage

```
gui/biomes
```

5.6.341 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`.

5.6.342 gui/choose-weapons

Tags: unavailable

Command: `gui/choose-weapons`

Ensure military dwarves choose appropriate weapons.

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
```

5.6.343 gui/civ-alert

Tags: fort | gameplay | interface | military | units

Command: `gui/civ-alert`

Quickly get your civilians to safety.

Normally, assigning a unit to a burrow is treated more like a suggestion than a command. This can be frustrating when you're assigning units to burrows in order to get them out of danger. In contrast, triggering a civilian alert with *gui/civ-alert* will cause all your non-military citizens to immediately rush to a burrow ASAP and stay there. This gives you a way to keep your civilians safe when there is danger about.

Usage

```
gui/civ-alert
```

How to set up and use a civilian alert

A civ alert needs a burrow to send civilians to. Go set one up if you haven't already. If you have walls around a secure interior, you can include all your below-ground area and the safe parts inside your walls. You can name the burrow "Inside" or "Safety" or "Panic room" or whatever you like.

Then, start up *gui/civ-alert* and select the burrow from the list. You can activate the civ alert right away with the button in the upper right corner. You can also access this button at any time from the squads panel.

When danger appears, open up the squads menu and click on the new "Activate civilian alert" button in the lower left corner. It's big and red; you can't miss it. Your civilians will rush off to safety and you can concentrate on dealing with the incursion without Urist McArmorsmith getting in the way.

When the civ alert is active, the civilian alert button will stay on the screen, even if the squads menu is closed. After the danger has passed, remember to turn the civ alert off again by clicking the button. Otherwise, your units will continue to be confined to their burrow and may eventually become unhappy or starve.

Overlay

The position of the "Activate civilian alert" button that appears when the squads panel is open is configurable via *gui/overlay*. The overlay panel also gives you a way to launch *gui/civ-alert* if you need to change which burrow civilians should be gathering at.

Technical notes

The functionality for civilian alerts is actually already inside the vanilla game. The ability to configure civilian alerts was lost when the DF UI was updated for the v50 release. This tool simply provides an interface layer for the vanilla functionality.

5.6.344 gui/clone-uniform

Tags: unavailable

Command: `gui/clone-uniform`

Duplicate an existing military uniform.

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
```

5.6.345 gui/color-schemes

Tags: unavailable

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 the fortress main map screen.

Usage

```
gui/color-schemes
```

5.6.346 gui/companion-order

Tags: unavailable

Command: `gui/companion-order`

Issue orders to companions.

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).

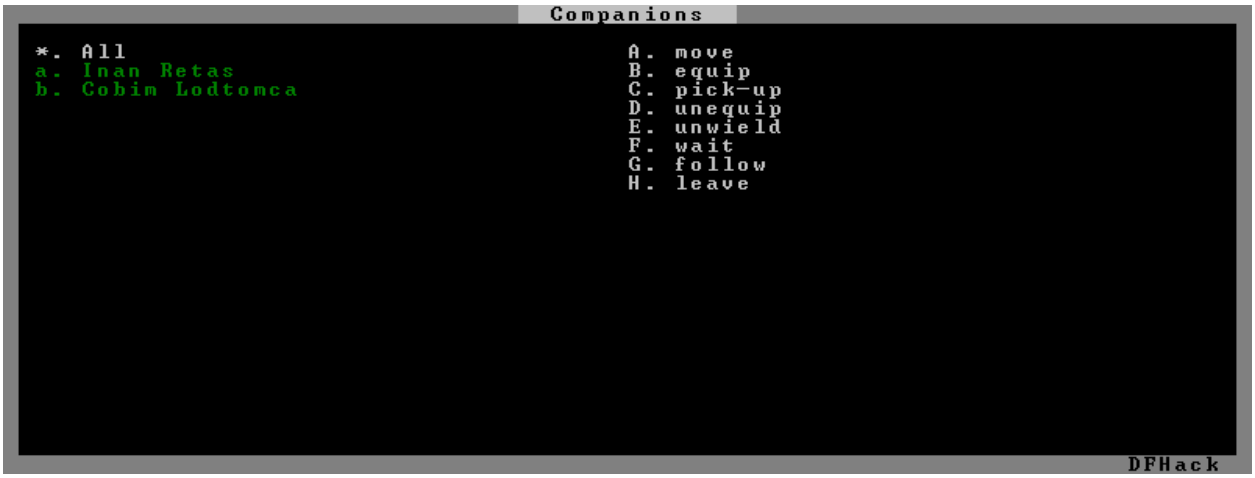
If `gui/companion-order` was called with the `-c` option, the following orders will be available:

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:



5.6.347 gui/confirm

Tags: fort | interface

Command: `gui/confirm`

Configure which confirmation dialogs are enabled.

This tool is a basic configuration interface for *confirm*. You can see and modify which confirmation dialogs are enabled. Your selections will be saved as personal preferences, and will apply to all forts going forward.

Usage

```
gui/confirm [<ID>]
```

You can optionally pass the ID of a confirmation dialog to have it initially selected in the list.

5.6.348 gui/control-panel

Tags: dfhack

Command: `gui/control-panel`

Configure DFHack and manage active DFHack tools.

Keybinding: CtrlShiftE

The DFHack control panel allows you to quickly see and change what DFHack tools are enabled, which tools will run when you start a new fort, which UI overlays are enabled, and how global DFHack configuration options are set. It also provides convenient links to relevant help pages and GUI configuration frontends (where available). The control panel has several sections that you can access by clicking on the tabs at the top of the window. Each tab has a search filter so you can quickly find the tools and options that you're looking for.

The tabs can also be navigated with the keyboard, with the `CtrlT` and `CtrlY` hotkeys. These are the default hotkeys for navigating DFHack tab bars.

The "Automation", "Bug Fixes", and "Gameplay" tabs

These three tabs provide access to the three main subcategories of DFHack tools. In general, you'll probably want to start with only the "Bugfix" tools enabled. As you become more comfortable with vanilla systems, and some of them start to become less fun and more toilsome, you can enable more of the "Automation" tools to manage them for you. Finally, you can examine the tools on the "Gameplay" tab and enable whatever you think sounds like fun :).

Under each of these tabs, there are two subtabs: "Enabled" and "Autostart". The subtabs can be navigated with the keyboard, using the `CtrlN` and `CtrlM` hotkeys.

The "Enabled" subtab

The "Enabled" tab allows you to toggle which tools are enabled right now. You can select the tool in the list to see a short description at the bottom. Hit `Enter`, double click on the tool name, or click on the toggle on the far left to enable or disable that tool.

Note that before a fort is loaded, there will be very few tools listed here. Come back when a fort is loaded to see much more.

Once tools are enabled, they will save their state with your fort and automatically re-enable themselves when you load that same fort again.

You can hit `CtrlH` or click on the help icon to show the help page for the selected tool in *gui/launcher*. You can also use this as shortcut to run custom commandline commands to configure that tool manually. If the tool has an associated GUI config screen, a gear icon will also appear next to the help icon. Hit `CtrlG`, click on the gear icon, or Shift-double click the tool name to launch the relevant configuration interface.

The "Autostart" subtab

This subtab is organized similarly to the "Enabled" subtab, but instead of tools you can enable now, it shows the tools that you can configure DFHack to auto-enable or auto-run when you start the game or a new fort. You'll recognize many tools from the "Enabled" subtab here, but there are also useful one-time commands that you might want to run at the start of a fort, like *ban-cooking all* or (if you have "mortal mode" disabled in the "Preferences" tab) god-mode tools like *light-aquifers-only*.

The "UI Overlays" tab

DFHack overlays add information and additional functionality to the vanilla DF screens. For example, the popular DFHack *Building Planner* is an overlay named `buildingplan.planner` that appears when you are building something.

The "Overlays" tab allows you to easily see which overlays are enabled, gives you a short description of what each one does, lets you toggle them on and off, and gives you links for the related help text (which is normally added at the bottom of the help page for the tool that provides the overlay). If you want to reposition any of the overlay widgets, hit `CtrlG` or click on the the hotkey hint to launch *gui/overlay*.

The "Preferences" tab

The "Preferences" tab allows you to change DFHack's internal settings and defaults, like whether DFHack's "mortal mode" is enabled – hiding the god-mode tools from the UI, whether DFHack tools pause the game when they come up, or how long you can take between clicks and still have it count as a double-click. Click on the gear icon or hit `Enter` to toggle or edit the selected preference.

Usage

```
gui/control-panel
```

5.6.349 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 in the parent window before starting this UI!

Usage

```
gui/cp437-table
```

5.6.350 gui/create-item

Tags: adventure | fort | armok | items

Command: `gui/create-item`

Summon items from the aether.

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 the quantity.

If a unit is selected, that unit will be designated the creator of the summoned items. Any item with a "sized for" property, like armor, will be created for that unit's race, and the items will appear at that unit's feet. If no unit is selected, the first citizen unit will be used as the creator.

Usage

```
gui/create-item [<options>]
```

Examples

gui/create-item
: Walk player through the creation of an item that can normally exist in the game.

gui/create-item --unrestricted --count 1
: Create one item made of anything in the game. For example, you can create a bar of vomit, if you please.

Options

-c, --count <num>
: Set the quantity of items to create instead of prompting for it.

-u, --unit <id>
: Use the specified unit as the "creator" of the generated item instead of the selected unit or the first citizen.

-f, --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 for a modded reaction with a code of `DFHACK_WISH`. When a reaction with that code is completed, show the item creation gui (with `--count 1`). This allows you to mod in "wands of wishing" that can let your adventurer make wishes for an item.

5.6.351 gui/design

Tags: fort | design | productivity | map

Command: `gui/design`

Design designation utility with shapes.

Keybinding: CtrlD in `dwarfmode/Default`

This tool provides a point and click interface to make designating shapes and patterns easier. Supports both digging designations and placing constructions.

Usage

```
gui/design
```

Overlay

This script provides an overlay that shows the selected dimensions when designating something with vanilla tools, for example when painting a burrow or designating digging. The dimensions show up in a tooltip that follows the mouse cursor.

5.6.352 gui/dfstatus

Tags: unavailable

Command: `gui/dfstatus`

Show a quick overview of critical stock quantities.

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
```

5.6.353 gui/embark-anywhere

Tags: embark | armok | interface

Command: `gui/embark-anywhere`

Embark wherever you want.

Keybinding: CtrlA in `choose_start_site`

If you run this command when you're choosing a site for embark, you can bypass any warnings the game gives you about potential embark locations. Want to embark in an inaccessible location on top of a mountain range? Go for it! Want to try a brief existence in the middle of the ocean? Nobody can stop you! Want to tempt fate by embarking *inside of* a necromancer tower? !!FUN!!

Any and all consequences of embarking in strange locations are up to you to handle (possibly with other armok tools).

Usage

```
gui/embark-anywhere
```

The command will only work when you are on the screen where you can choose your embark site. The DFHack logo is not displayed on that screen since its default position conflicts with the vanilla embark size selection panel. Remember that you can bring up DFHack's *context menu* with `CtrlShiftC` or the *in-game command launcher* directly with the backtick key (`).

5.6.354 gui/extended-status

Tags: unavailable

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.

5.6.355 gui/family-affairs

Tags: unavailable

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.

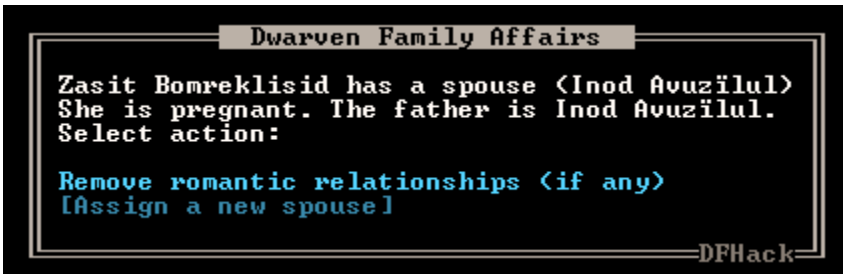
Usage

`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



5.6.356 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.

Select a field and hit `Enter` or double click to edit, or, for structured fields, hit `Enter` or single click to inspect their contents. Right click or hit `Esc` to go back to the previous structure you were inspecting. Right clicking when viewing the structure you started with will exit the tool. Hold down `Shift` and right click to exit, even if you are inspecting a substructure, no matter how deep.

If you just want to browse without fear of accidentally changing anything, hit `CtrlD` to toggle read-only mode. If you want *gui/gm-editor* to automatically pick up changes to game data in realtime, hit `AltA` to switch to auto update mode.

Warning: Note that data structures can be created and deleted while the game is running. If you happen to be inspecting a dynamically allocated data structure when it is deleted by the game, the game may crash. Please save your game before poking around in *gui/gm-editor*, especially if you are examining data while the game is unpaused.

Usage

`gui/gm-editor [-f]`
Open the editor on whatever is selected or viewed (e.g. unit/item/building/ engraving/etc.)

`gui/gm-editor [-f] <lua expression>`
Evaluate a lua expression and opens the editor on its results. Field prefixes of `df.global` can be omitted.

`gui/gm-editor [-f] dialog`
Show an in-game dialog to input the lua expression to evaluate. Works the same as the version above.

Examples

`gui/gm-editor`
Opens the editor on the selected unit/item/job/workorder/stockpile etc.

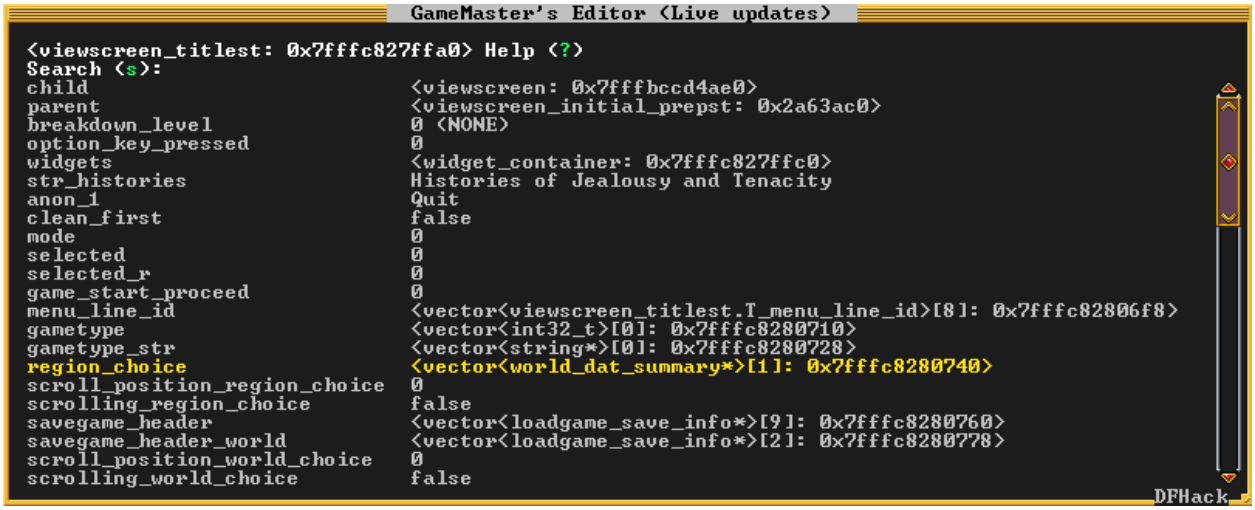
`gui/gm-editor world.items.all`
Opens the editor on the items list.

`gui/gm-editor --freeze scr`
Opens the editor on the current DF viewscreen data (bypassing any DFHack layers) and prevents the underlying viewscreen from getting updates while you have the editor open.

Options

-f, --freeze
: Freeze the underlying viewscreen so that it does not receive any updates. This allows you to be sure that whatever you are inspecting or modifying will not be read or changed by the game until you are done with it. Note that this will also prevent any rendering refreshes, so the background is replaced with a blank screen. You can open multiple instances of *gui/gm-editor* as usual when the game is frozen. The black background will disappear when the last *gui/gm-editor* window that was opened with the `--freeze` option is dismissed.

Screenshot



5.6.357 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]
```

5.6.358 gui/guide-path

Tags: unavailable

Command: `gui/guide-path`

Visualize minecart guide paths.

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

5.6.359 gui/kitchen-info

Tags: unavailable

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
```

5.6.360 gui/launcher

Tags: dfhack

Command: `gui/launcher`

In-game DFHack command launcher with integrated help.

Keybinding: `

Keybinding: CtrlShiftD

Keybinding: CtrlShiftP -> `"gui/launcher --minimal"`

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]
gui/launcher -m|--minimal [initial commandline]
```

Examples

`gui/launcher`
: Open the launcher dialog with a blank initial commandline.

`gui/launcher --minimal`
: Open the launcher dialog in minimal mode 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 lower panel.

Editing and running commands

Enter the command you want to run by typing its name. If you want to start over, `CtrlX` 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 lower panel after you run it. If you want to run the command but close the dialog immediately so you can get back to the game, 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 (the separate window that comes up when you start DF) if you need to find it later.

To pause or unpause the game while *gui/launcher* is open, hit the spacebar once or twice. If you are typing a command, the first space will go into the edit box for your commandline. If the commandline is empty or if it already ends in a space, the space key will be passed through to the game to affect the pause button.

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 text you need by clicking on the characters with the mouse and then clicking the `Enter` button to send the text to the launcher editor.

Autocomplete

As you type, autocomplete options for DFHack commands appear in the right column. You can restrict which commands are shown in the autocomplete list by setting the tag filter with `CtrlW` or by clicking on the `Tags` button. If the first word of what you've typed matches a valid command, then the autocomplete options switch to showing commands that have similar functionality to the one that you've typed. Click on an autocomplete list option to select it or cycle through them with `Tab` and `ShiftTab`. You can run a command quickly without parameters by double-clicking on the tool name in the list. Holding down shift while you double-click allows you to run the command and close *gui/launcher* at the same time.

Context-sensitive help and command output

When you start `gui/launcher` without parameters, it shows some useful information in the lower panel about how to get started with DFHack.

Once you have typed (or autocompleted) a word that matches a valid command, the lower panel shows the help for that command, including usage instructions and examples. You can scroll the help text with the mouse wheel or with `PgUp` and `PgDn`. You can also scroll line by line with `ShiftUp` and `ShiftDown`.

Once you run a command, the lower panel will switch to command output mode, where you can see any text the command printed to the screen. If you want to see more help text as you browse further commands, you can switch the lower panel back to help mode with `CtrlT`. The output text is kept for all the commands you run while the launcher window is open (up to 256KB of text), but only the most recent 32KB of text is saved if you dismiss the launcher window and bring it back up. Command output is also printed to the external DFHack console (the one you can show with *show* on Windows) or the parent terminal on Unix-based systems if you need a longer history of the output.

You can run the *clear* command or click the `Clear output` button to clear the output scrollback buffer.

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` arrow keys. You can also 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 hit `Esc` to edit the command before running it.

Default tag filters

By default, commands intended for developers and modders are filtered out of the autocomplete list. This includes any tools tagged with `unavailable`. If you have "mortal mode" enabled in the *gui/control-panel* preferences, any tools with the `armok` tag are filterd out as well.

You can toggle this default filtering by hitting `CtrlD` to switch into "Dev mode" at any time. You can also adjust your command filters in the `Tags` filter list.

5.6.361 gui/liquids

Tags: fort | armok | map

Command: `gui/liquids`

Interactively paint liquids onto the map.

This tool is a gui for spawning water, magma, and river sources on the map, allowing you to click and paint liquid onto the map. It can optionally clean existing liquids on the map from salt or being stagnant.

Warning: There is **no undo support**. Be sure you know what you are doing before spawning any liquid, especially river sources!

Usage

```
gui/liquids
```

5.6.362 gui/load-screen

Tags: unavailable

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
```

5.6.363 gui/mass-remove

Tags: fort | design | productivity | buildings | stockpiles

Command: `gui/mass-remove`

Mass select things to remove.

Keybinding: CtrlM in `dwarfmode/Default`

This tool lets you remove buildings, constructions, stockpiles, and/or zones using a mouse-driven box selection. You can choose which you want to remove with the given filters, then box select to apply to the map.

Planned buildings, constructions, stockpiles, and zones will be removed immediately. Built buildings and constructions will be designated for deconstruction.

Usage

```
gui/mass-remove
```

5.6.364 gui/masspit

Tags: fort | productivity | animals

Command: `gui/masspit`

Designate creatures for pitting.

Using this tool you can designate all caged creatures in an animal stockpile to be sent to a pit.

The script provides a visual interface, you can select the stockpile/pit using keyboard controls or click the map.

Usage

```
gui/masspit
```

5.6.365 gui/mod-manager

Tags: dfhack | interface

Command: `gui/mod-manager`

Save and restore lists of active mods.

Adds an optional overlay to the mod list screen that allows you to save and load mod list presets, as well as set a default mod list preset for new worlds.

Usage

```
gui/mod-manager
```

5.6.366 gui/notify

Tags: fort | interface

Command: `gui/notify`

Show notifications for important events.

This tool is the configuration interface for the provided overlay. It allows you to select which notifications to enable for the overlay display. See the descriptions in the *gui/notify* list for more details on what each notification is for.

Usage

```
gui/notify
```

Overlay

This script provides an overlay that shows the currently enabled notifications (when applicable). If you click on an active notification in the list, it will bring up a screen where you can do something about it or will zoom the map to the target. If there are multiple targets, each successive click on the notification (or press of the `Enter` key) will zoom to the next target. You can also shift click (or press `ShiftEnter`) to zoom to the previous target.

5.6.367 gui/overlay

Tags: dfhack | interface

Command: `gui/overlay`

Reposition DFHack overlay widgets.

Whereas *gui/control-panel* provides a way to browse installed overlays, enable them, and see their help, this tool allows you to reposition the *overlay* widgets to your liking. You can either see all overlay widgets or just the ones configured to appear on the current DF screen. Each visible overlay widget will be highlighted in a frame so you can find them easily. You can click on the frame and drag the widget around the screen to reposition it, or hit `Enter` with the widget selected and reposition with the cursor keys.

The frame around the selected widget will show a yellow 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
```

5.6.368 gui/pathable

Tags: fort | inspection | map

Command: `gui/pathable`

Highlights tiles reachable from the selected tile.

This tool highlights each visible map tile to indicate whether it is possible to path to that tile from the tile under the mouse cursor. You can move the mouse (and the map) around and the highlight will change dynamically.

If graphics are enabled, then tiles show a small yellow box if they are pathable and a small black box if not.

In ASCII mode, the tiles are highlighted in green if pathing is possible and red if not.

While the UI is active, you can use the following hotkeys to change the behavior:

- `Ctrlt`: Lock target: when enabled, you can move the map around and the target tile will not change. This is useful to check whether parts of the map far away from the target tile can be pathed to from the target tile.

- `Ctrld`: Draw: allows temporarily disabling the highlighting entirely. This allows you to see the map without the highlights, if desired.
- `Ctrlu`: Skip unrevealed: when enabled, unrevealed tiles will not be highlighted at all instead of being highlighted as not pathable. This might be useful to turn off if you want to see the pathability of unrevealed cavern sections.

You can drag the informational panel around while it is visible if it's in the way.

Note: This tool uses a cache used by DF, which currently does *not* account for climbing or flying. If an area of the map is only accessible by climbing or flying, this tool may report it as inaccessible. Care should be taken when digging into the upper levels of caverns, for example.

Usage

```
gui/pathable
```

5.6.369 gui/petitions

Tags: fort | inspection

Command: `gui/petitions`

List guildhall and temple agreements.

Show your fort's petitions, both pending and fulfilled.

Usage

```
gui/petitions
```

5.6.370 gui/power-meter

Tags: unavailable

Command: `gui/power-meter`

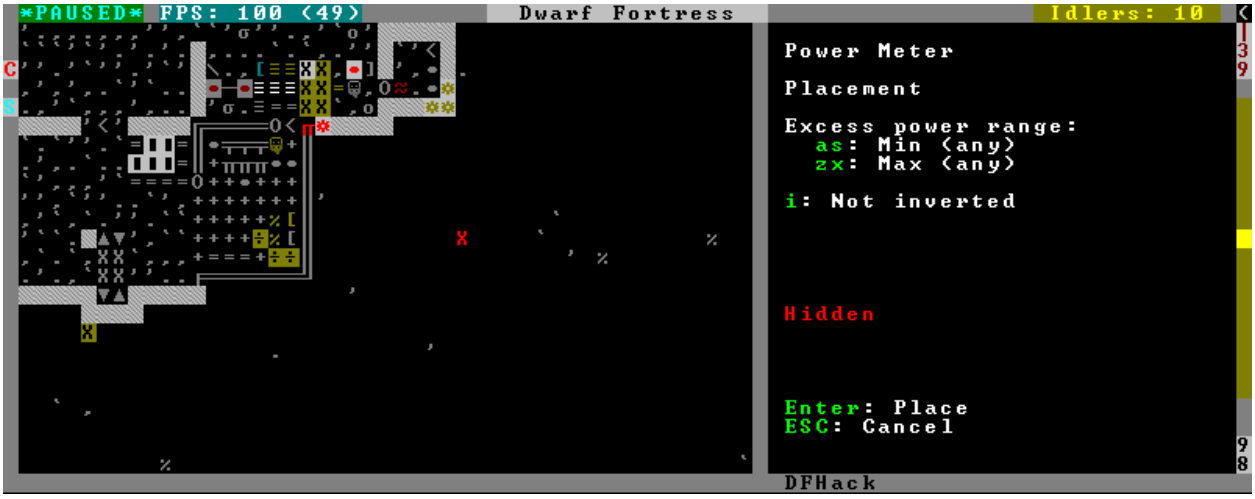
Allow pressure plates to measure power.

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



5.6.371 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.

5.6.372 gui/quantum

Tags: fort | productivity | map | 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 a large 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 on or more "feeder" stockpiles to a one-tile minecart hauling route. As soon as an item from the feeder stockpile(s) 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 holds all the items and is your quantum stockpile. You can also choose to not have a receiving stockpile and instead have the minecart dump into a pit (perhaps a pit filled with magma).

Before you run this tool, create and configure your "feeder" stockpile(s). The size of the feeders determine 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 a feeder stockpile by clicking on it. If you want to select multiple feeder stockpiles, switch the feeder selection toggle into multi mode.
2. Set configuration with the onscreen options.
3. Click on the map to build the quantum stockpile there.

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 a wooden one for you. Once it is built, you'll have to run *assign-minecarts all* to assign it to the route or open the (H)auling menu and assign it manually. The quantum stockpile will not function until the minecart is in place.

See the wiki for more information on quantum stockpiles.

Usage

```
gui/quantum
```

Tips

Loading items into minecarts is a low priority task. If you find that your feeder stockpiles are filling up because your dwarves aren't loading the items into the minecarts, there are a few things you could change to get things moving along:

- Make your dwarves less busy overall by reducing the number of other jobs they have to do
- Assign a few dwarves the Hauling work detail and specialize them so they focus on those tasks. Note that there is no specific labor for loading items into vehicles, it's just "hauling" in general.

- Run `prioritize -a StoreItemInVehicle`, which causes the game to prioritize the minecart loading tasks. Note that this can pull artisans away from their workshops to go load minecarts. You can protect against this by specializing your artisans who are assigned to workshops.

5.6.373 gui/quickcmd

Tags: dfhack

Command: `gui/quickcmd`

Quickly run saved commands.

Keybinding: CtrlShiftA

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
```

5.6.374 gui/quickfort

Tags: fort | design | productivity | buildings | map | stockpiles

Command: `gui/quickfort`

Apply layout blueprints to your fort.

Keybinding: CtrlShiftQ in `dwarfmode`

This is the graphical interface for the *quickfort* script. Once you load a blueprint, you will see a highlight over the tiles that will be modified. You can use the mouse cursor to reposition the blueprint and the hotkeys to rotate and repeat the blueprint up or down z-levels. Once you are satisfied, click the mouse or hit `Enter` to apply the blueprint to the map.

You can apply the blueprint as many times as you wish to different spots on the map. If a blueprint that you designated was only partially applied (due to job cancellations, incomplete dig area, or any other reason) you can apply the blueprint a second time in the same spot to fill in any gaps. Any part of the blueprint that has already been completed will be harmlessly skipped. Right click or hit `Esc` to close the *gui/quickfort* UI.

Note that *quickfort* blueprints will use the DFHack building planner (*buildingplan*) material filter settings. If you want specific materials to be used, use the building planner UI to set the appropriate filters before applying a blueprint.

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 filters, the id numbers in the list may not be contiguous.

To rotate or flip the blueprint around, enable transformations with `t` and use the following keys to add a transformation step:

(: Rotate counterclockwise (ccw)): Rotate clockwise (cw) _: Flip vertically (vflip) =: 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.

5.6.375 gui/rename

Tags: unavailable

Command: `gui/rename`

Give buildings and units new names, optionally with special chars.

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

5.6.376 gui/reveal

Tags: adventure | fort | armok | map

Command: `gui/reveal`

Reveal map tiles.

This script provides a means for you to safely glimpse at unexplored areas of the map, such as aquifers or the caverns, so you can plan your fort with full knowledge of the terrain. When you open *gui/reveal*, the map will be revealed. You can see where the caverns are, and you can designate what you want for digging. When you close *gui/reveal*, the map will automatically be unrevealed so you can continue normal gameplay. If you want the reveal to be permanent, you can toggle the setting before you close *gui/reveal*.

You can choose to only reveal the aquifers and not other tiles by toggling the settings in the UI or by specifying the appropriate commandline parameter when starting *gui/reveal*.

Areas with event triggers, such as gem boxes and adamantine spires, are not revealed by default. This allows you to choose to keep the map unrevealed when you close the *gui/reveal* UI without being immediately inundated with thousands of event message popups.

In graphics mode, solid tiles that are not adjacent to open space will not be rendered, but they can still be examined by hovering over them with the mouse. Switching to ASCII mode (in the game settings) will allow the display of the revealed tiles, allowing you to quickly determine where the ores and gem clusters are.

Usage

```
gui/reveal [hell] [<options>]
```

Pass the `hell` keyword to fully reveal adamantine spires, gemstone pillars, and the underworld. The game cannot be unpaused with these features revealed, so the choice to keep the map unrevealed when you close *gui/reveal* is disabled when this option is specified.

Examples

`gui/reveal`
: Reveal all "normal" terrain, but keep areas with late-game surprises hidden.

`gui/reveal hell`
: Fully reveal adamantine spires, gemstone pillars, and the underworld. The game cannot be unpaused with these features revealed, so the choice to keep the map unrevealed when you close *gui/reveal* is disabled when this option is specified.

Options

`-o`, `--aquifers-only`
: Don't reveal any map tiles, but continue to display markers to identify aquifers and damp tiles as per the *dig.warmdamp* overlay.

5.6.377 gui/room-list

Tags: unavailable

Command: `gui/room-list`

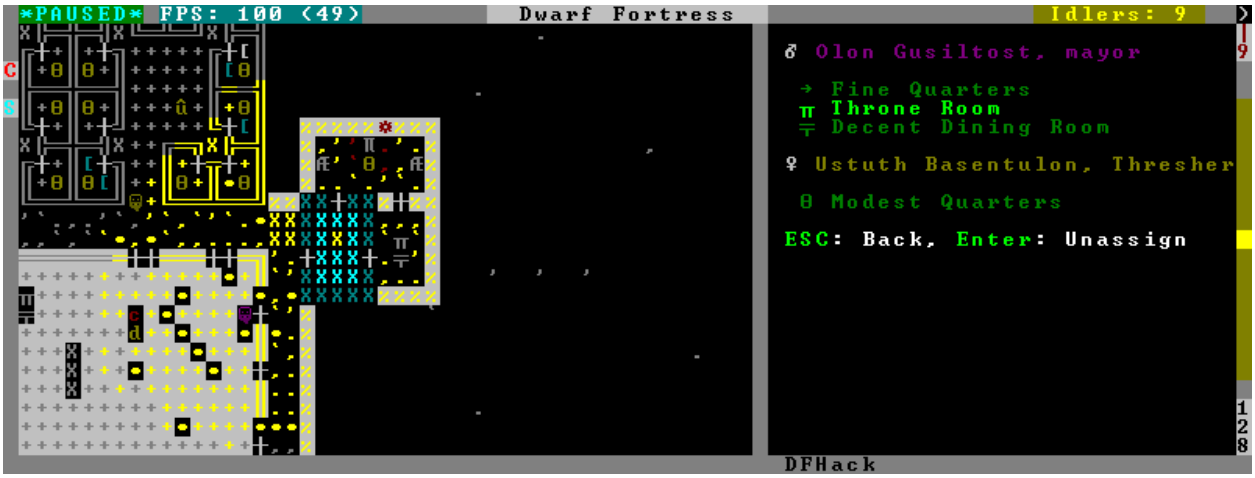
Manage rooms owned by a dwarf.

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



5.6.378 gui/sandbox

Tags: adventure | fort | armok | animals | items | map | plants | units

Command: `gui/sandbox`

Create units, trees, or items.

This tool provides a spawning interface for units, trees, and/or items. Units can be created with arbitrary skillsets, and trees can be created either as saplings or as fully grown (depending on the age you set). The item creation interface is the same as *gui/create-item*.

You can choose whether spawned units are:

- hostile (default)
- hostile undead
- independent/wild
- friendly
- citizens/pets

Note that if you create new citizens and you're not using *autolabor*, you'll have to got into the labors screen and make at least one change (any change) to get DF to assign them labors. Otherwise they'll stand around with "No job".

Usage

```
gui/sandbox
```

Caveats

If running from adventure mode, the map will show fort-mode "dig" markers on tiles that were within the code of vision of your adventurers. This is visually distracting, but it is not an error and can be ignored.

5.6.379 gui/seedwatch

Tags: fort | auto | plants

Command: `gui/seedwatch`

Manages seed and plant cooking based on seed stock levels.

This is the configuration interface for the *seedwatch* plugin. You can configure a target stock amount for each seed type. If the number of seeds of that type falls below the target, then the plants and seeds of that type will be protected from cookery. If the number rises above the target + 20, then cooking will be allowed again.

Usage

```
gui/seedwatch
```

5.6.380 gui/settings-manager

Tags: embark | interface

Command: `gui/settings-manager`

Import and export DF settings.

This tool allows you to save and load DF settings.

Usage

```
gui/settings-manager save-difficulty
gui/settings-manager load-difficulty
gui/settings-manager save-standing-orders
gui/settings-manager load-standing-orders
gui/settings-manager save-work-details
gui/settings-manager load-work-details
```

Difficulty can be saved and loaded on the embark "preparation" screen or in an active fort. Standing orders and work details can only be saved and loaded in an active fort.

If auto-restoring of difficulty settings is turned on, it happens when the embark "preparation" screen is loaded. If auto-restoring of standing orders or work details definitions is turned on, it happens when the fort is loaded for the first time (just like all other Autostart commands configured in *gui/control-panel*).

Overlays

When embarking or when a fort is loaded, if you click on the `Custom settings` button for game difficulty, you will see a new panel at the top. You can save the current difficulty settings and load the saved settings back. You can also toggle an option to automatically load the saved settings for new embarks.

When a fort is loaded, you can also go to the Labor -> Standing Orders page. You will see a new panel that allows you to save and restore your settings for standing orders. You can also toggle whether the saved standing orders are automatically restored when you embark on a new fort. This will toggle the relevant command in *gui/control-panel* on the Automation -> Autostart page.

There is a similar panel on the Labor -> Work Details page that allows for saving and restoring of work detail definitons. Be aware that work detail assignments to units cannot be saved, so you have to assign the work details to individual units after you restore the definitions. Another caveat is that DF doesn't evaluate work detail definitions until a change (any change) is made on the work details screen. Therefore, after importing work detail definitions, including auto-loading them for new embarks, you have to go to the work details page and make a change before your imported work details will take effect.

5.6.381 gui/siege-engine

Tags: unavailable

Command: `gui/siege-engine`

Extend the functionality and usability of siege engines.

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.

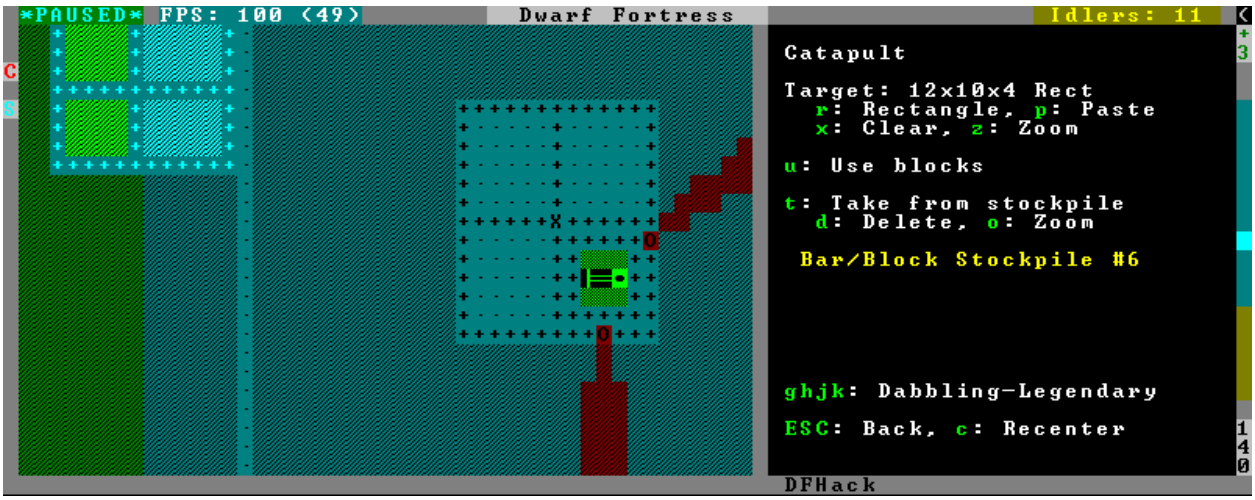
Pressing `t` switches to a mode for selecting a stockpile to take ammo from.

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



5.6.382 gui/stamper

Tags: unavailable

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
```

5.6.383 gui/stockpiles

Tags: unavailable

Command: `gui/stockpiles`

Import and export stockpile settings.

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
```

5.6.384 gui/suspendmanager

Tags: fort | jobs

Command: `gui/suspendmanager`

Intelligently suspend and unsuspend jobs.

This is the graphical configuration interface for the *suspendmanager* automation tool.

Usage

```
gui/suspendmanager
```

5.6.385 gui/teleport

Tags: fort | armok | units

Command: `gui/teleport`

Teleport units anywhere.

Keybinding: CtrlShiftT in `dwarfmode`

This tool allows you to interactively select units to teleport by drawing boxes around them on the map. Double clicking on a destination tile will teleport the selected units there.

If a unit is already selected in the UI when you run *gui/teleport*, it will be pre-selected for teleport.

Note that you *can* select enemies that are lying in ambush and are not visible on the map yet, so you if you select an area and see a marker that indicates that a unit is selected, but you don't see the unit itself, this is likely what it is. You can stil teleport these units while they are hidden.

Usage

```
gui/teleport
```

5.6.386 gui/unit-info-viewer

Tags: adventure | fort | inspection | units

Command: `gui/unit-info-viewer`

Display detailed information about a unit.

Displays information about age, birth, maxage, shearing, milking, grazing, egg laying, body size, and death for the selected unit.

You can click on different units while the tool window is open and the displayed information will refresh for the selected unit.

Usage

```
gui/unit-info-viewer
```

5.6.387 gui/unit-syndromes

Tags: fort | inspection | units

Command: `gui/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

```
gui/unit-syndromes
```

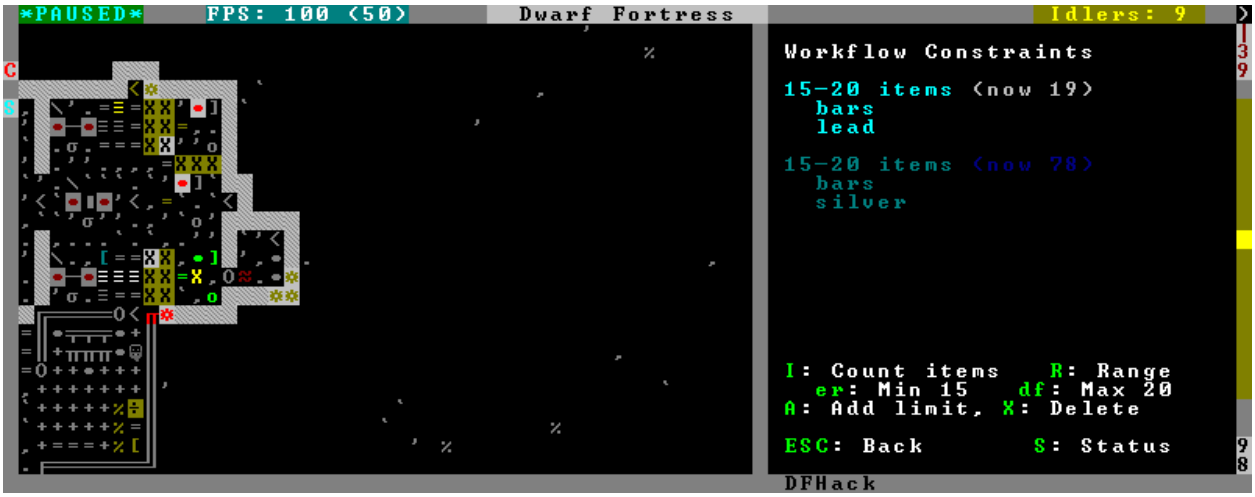
5.6.388 gui/workflow

Tags: unavailable

Command: `gui/workflow`

Manage automated item production rules.

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.

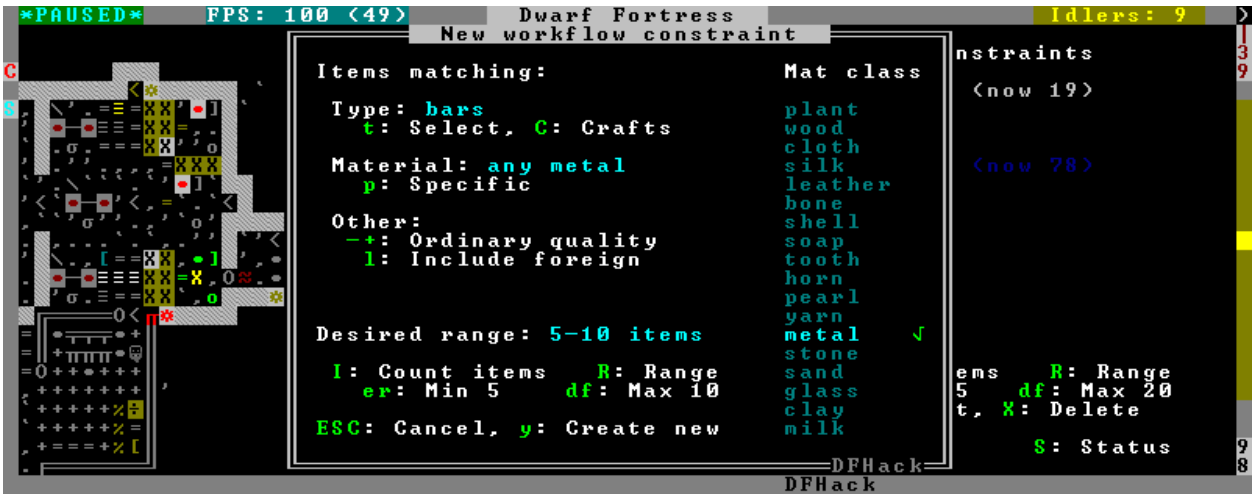


A constraint is a target range to be compared against either the number of individual items or the number of item stacks. It also includes an item type and, optionally, a material. When the current stock count is below the lower bound of the range, the job is resumed; if it is above or equal to the top 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.

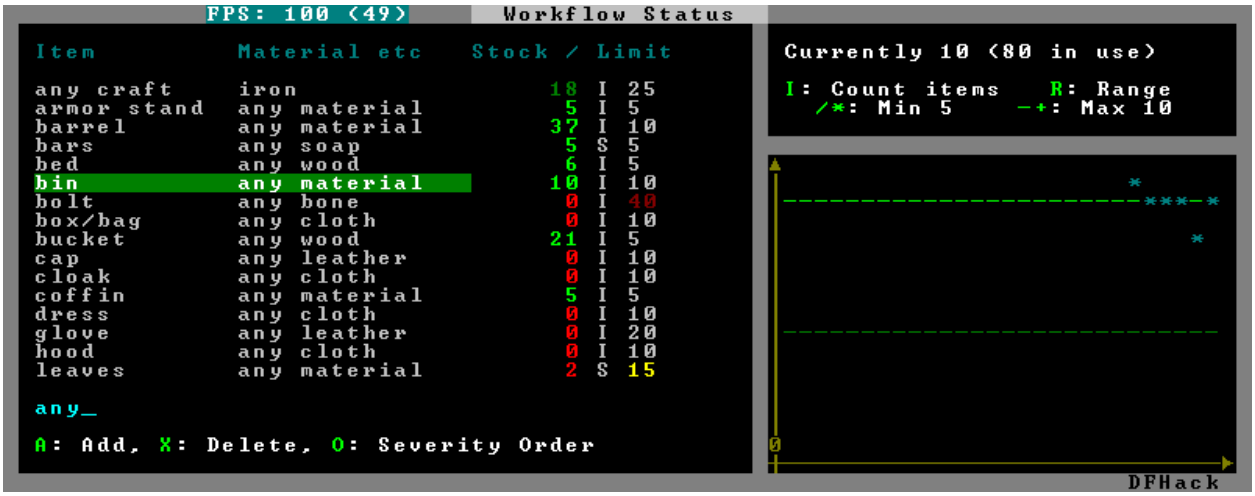
Pressing `I` switches the current constraint between counting stacks and counting individual items. Pressing `R` lets you input the range directly, or `e`, `r`, `d`, `f` incrementally adjusts the bounds.

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.

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.



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:



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.

5.6.389 gui/workorder-details

Tags: unavailable | fort | inspection | workorders

Command: `gui/workorder-details`

Adjust input materials and traits for workorders.

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
```

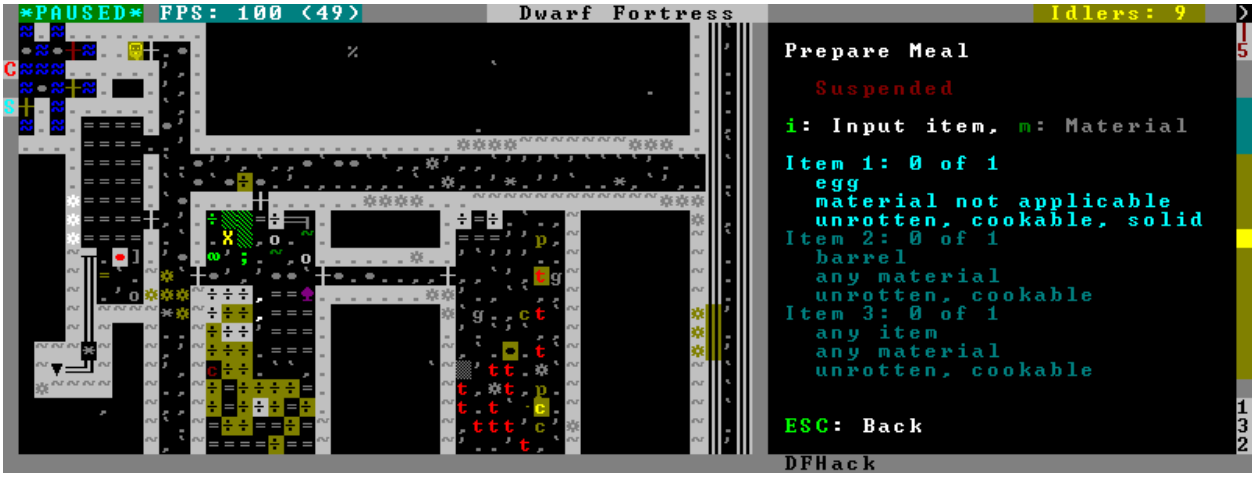
5.6.390 gui/workshop-job

Tags: unavailable | fort | inspection | jobs

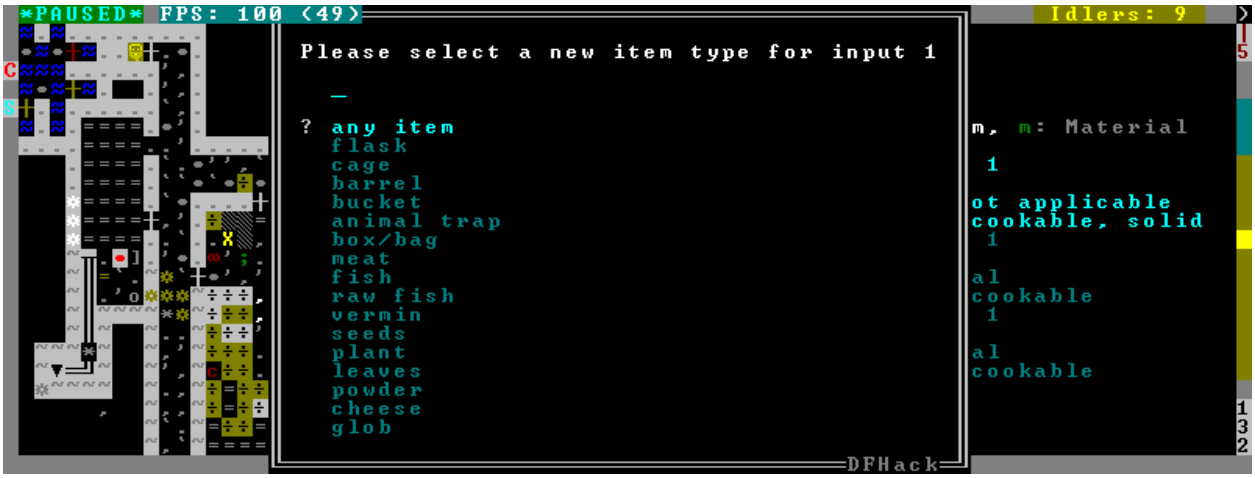
Command: `gui/workshop-job`

Adjust the input materials used for a job at a workshop.

This tool allows you to inspect or change the input reagents for the selected workshop job (in `q` mode).



Pressing `i` shows a dialog where you can select an item type from a list.



Pressing `m` (unless the item type does not allow a material) lets you choose a material.

Since there are a lot more materials than item types, this dialog is more complex and uses a hierarchy of sub-menus. List choices that open a sub-menu are marked with an arrow on the left.

Warning: Due to the way input reagent matching works in DF, you must select an item type if you select a material or the material 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
```

5.6.391 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.

5.6.392 modtools/anonymous-script

Tags: unavailable

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"
```

5.6.393 modtools/change-build-menu

Tags: unavailable

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.

5.6.394 modtools/create-item

Tags: dev

Command: `modtools/create-item`

Create arbitrary items.

This tool provides a commandline interface for creating items of your choice.

Usage

```
modtools/create-item <options>
```

Examples

`modtools/create-item -u 23145 -i WEAPON:ITEM_WEAPON_PICK -m INORGANIC:IRON -q4`
Have unit 23145 create an exceptionally crafted iron pick.

`modtools/create-item -u 323 -i MEAT:NONE -m CREATURE:DWARF:BRAIN`
Have unit 323 produce a lump of (prepared) brain.

`modtools/create-item -i BOULDER:NONE -m INORGANIC:RAW_ADAMANTINE -c 5`
Spawn 5 raw adamantine boulders.

`modtools/create-item -i DRINK:NONE -m PLANT:MUSHROOM_HELMET_PLUMP:DRINK`
Spawn a barrel of dwarven ale.

Options

`-u`, `--unit <id>` (default: first citizen)
The ID of the unit to use as the item's creator. You can also pass the string "\LAST" to use the most recently created unit.

`-i`, `--item <itemdef>` (required)
The def string of the item you want to create.

`-m`, `--material <matdef>` (required)
That def string of the material you want the item to be made out of.

`-q`, `--quality <num>` (default: `0`, equal to `df.item_quality.Ordinary`)
The quality of the created item.

`-d`, `--description <string>` (required if you are creating a slab)
The text that will be engraved on the created slab.

`-c`, `--count <num>` (default: `1`)
The number of items to create. If the item is stackable, this will be the stack size.

`-t`, `--caste <name or num>` (default: `0`)
Used if producing a corpse or other creature-based item that could have a caste associated with it.

`-p`, `--pos <x>,<y>,<z>`
If specified, items will be spawned at the given coordinates instead of at the creator unit's feet.

5.6.395 modtools/create-tree

Tags: unavailable

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 ]
```

5.6.396 modtools/create-unit

Tags: unavailable

Command: `modtools/create-unit`

Create arbitrary units.

Creates a unit.

Usage

```
-race raceName
    (obligatory)
    Specify the race of the unit to be created.
    examples:
        DWARF
        HUMAN

-caste casteName
    Specify the caste of the unit to be created.
    If omitted, the caste is randomly selected.
    examples:
        MALE
        FEMALE
        DEFAULT

-domesticate
    Tames the unit if it lacks the CAN_LEARN and CAN_SPEAK tokens.

-civId id
    Make the created unit a member of the specified civilisation
    (or none if id = -1).  If id is \\LOCAL, make it a member of the
    civ associated with the fort; otherwise id must be an integer

-groupId id
    Make the created unit a member of the specified group
    (or none if id = -1).  If id is \\LOCAL, make it a member of the
    group associated with the fort; otherwise id must be an integer

-setUnitToFort
    Sets the groupId and civId to those of the player in Fortress mode.
    Equivalent to -civId \\LOCAL and -groupId \\LOCAL.

-name entityRawName
    Set the unit's name to be a random name appropriate for the
    given entity. \\LOCAL can be specified instead to automatically
    use the fort group entity in fortress mode only. 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.
```

(continues on next page)

(continued from previous page)

```
-equip [ ITEM:MATERIAL:QUANTITY ... ]
    This can be included to create items and equip them onto
        the created unit.
    This is carried out via the same logic used in arena mode,
        so equipment will always be sized correctly and placed
        on what the game deems to be appropriate bodyparts.
        Clothing is also layered in the appropriate order.
    Note that this currently comes with some limitations,
        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
```

(continues on next page)

(continued from previous page)

```
    This can be included to give the unit a custom profession name.
    Enclose the name in quotation marks if it includes spaces.
    example:
        -customProfession "Destroyer of Worlds"

-duration ticks
    If this is included, the unit will vanish in a puff of smoke
        once the specified number of ticks has elapsed.
    Replace "ticks" with an integer greater than 0.
    Note that the unit's equipment will not vanish.

-quantity howMany
    This can be included to create multiple creatures simultaneously.
    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
```

(continues on next page)

(continued from previous page)

```
        df.unit_flags2
        df.unit_flags3
        df.unit_flags4
    example:
        flagSet [ announce_titan ]
            causes an announcement describing the unit to appear
            when it is discovered ("[Unit] has come! ...")

-flagClear [ flag1 flag2 ... ]
    As above, but sets the specified unit flags to false.
```

5.6.397 modtools/equip-item

Tags: unavailable

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*.

5.6.398 modtools/extra-gamelog

Tags: unavailable

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
```

5.6.399 modtools/fire-rate

Tags: unavailable

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!**

5.6.400 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
```

5.6.401 modtools/if-entity

Tags: dev

Command: `modtools/if-entity`

Run DFHack commands based on the the civ id of the current fort.

Run a command if the current fort entity matches a given ID.

This script can only be called when a fort is loaded. To run it immediately when a matching fort is loaded, call it from a registered `dfhack.onStateChange` *state change handler*. See the *DFHack modding guide* for an example of how to set up a state change handler.

Usage

```
modtools/if-entity --id <entity id> --cmd [ <command> ]
```

Options

--id <entity id>
Specify the entity ID to match.

--cmd [<command>]
Specify the command to be run when the given id is matched.

Example

modtools/if-entity --id FOREST --cmd [lua "print('Dirty hippies.')"]
Print a message if you load an elf fort, but not a dwarf, human, etc. fort.

5.6.402 modtools/interaction-trigger

Tags: unavailable

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
```

(continues on next page)

(continued from previous page)

```
-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.

5.6.403 modtools/invader-item-destroyer

Tags: unavailable

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
```

5.6.404 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 or unequips items or attacks another unit with specified item types, specified item materials, or specified item contaminants.

Usage

```
modtools/item-trigger [<options>] --command [ <command> ]
```

At least one of the following options must be specified when registering a trigger: `--itemType`, `--material`, or `--contaminant`.

Options

--clear
Clear existing registered triggers before adding the specified trigger. If no new trigger is specified, this option just clears existing triggers.

--checkAttackEvery <n>
Check for attack events at least once every n ticks.

--checkInventoryEvery <n>
Check for inventory events at least once every n ticks.

--itemType <type>
Trigger the command for items of this type (as specified in the raws). Examples:

```
ITEM_WEAPON_PICK
RING
```

--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
```

--onStrike
Trigger the command on appropriate weapon strikes.

--onEquip <mode>
Trigger the command when someone equips an appropriate item. Optionally, the equipment mode can be specified. Possible values for mode:

```
Hauled
Weapon
Worn
Piercing
Flask
WrappedAround
StuckIn
InMouth
Pet
SewnInto
Strapped
```

Multiple values can be specified simultaneously. Example:

```
-onEquip [ Weapon Worn Hauled ]
```

--onUnequip <mode>
Trigger the command when someone unequips an appropriate item. Same mode values as `--onEquip`.

--command [<commandStrs>]
Specify the command to be executed. The following tokens can be used in the command and they will be replaced with appropriate values:

```
\\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
```

5.6.405 modtools/moddable-gods

Tags: unavailable

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
```

5.6.406 modtools/outside-only

Tags: unavailable

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
```

5.6.407 modtools/pref-edit

Tags: unavailable

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
```

5.6.408 modtools/projectile-trigger

Tags: unavailable

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
```

5.6.409 modtools/random-trigger

Tags: unavailable

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.

5.6.410 modtools/raw-lint

Tags: unavailable

Command: `modtools/raw-lint`

Check for errors in raw files.

Checks for simple issues with raw files. Can be run automatically.

5.6.411 modtools/reaction-product-trigger

Tags: unavailable

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
```

5.6.412 modtools/reaction-trigger

Tags: unavailable

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
    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
```

(continues on next page)

(continued from previous page)

```
    when used with -syndrome, the target must be valid for the syndrome
    otherwise, the command will not be run for that target
```

5.6.413 modtools/reaction-trigger-transition

Tags: unavailable

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.

5.6.414 modtools/set-belief

Tags: unavailable

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 *.

5.6.415 modtools/set-need

Tags: unavailable

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
```

5.6.416 modtools/set-personality

Tags: unavailable

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> [<other␣
↪options>]
```

If no target option is given, the unit selected in the UI is used by default.

Target options

--citizens
: All citizens and residents 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.

5.6.417 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

5.6.418 modtools/spawn-flow

Tags: unavailable

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).

5.6.419 modtools/spawn-liquid

Tags: dev

Command: `modtools/spawn-liquid`

Spawn a liquid at a given position.

This script spawns liquid at the given coordinates.

Usage

```
modtools/spawn-liquid --type <type> --level <level> --position <x>,<y>,<z>
```

Options

`--type <type>`

Liquid tile type:
Water Magma

`--level <level>`
The amount of liquid units to spawn from 1-7.

`--position <x>,<y>,<z>`
The position at which to spawn the liquid.

Examples

`modtools/spawn-liquid --type Water --level 7 --position 60,60,143`
Spawn 7/7 Water on tile coordinates 60, 60, 143

5.6.420 modtools/syndrome-trigger

Tags: unavailable

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 ] ]
```

5.6.421 modtools/transform-unit

Tags: unavailable

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

SIX

USER GUIDES

These pages are detailed guides covering DFHack tools.

6.1 DFHack modding guide

6.1.1 What is the difference between a script and a mod?

Well, sometimes there is no difference. A mod is anything you add to the game, which can be graphics overrides, content in the raws, DFHack scripts, any, or all. There are already resources out there for raws modding, so this guide will focus more on scripts, both standalone and as an extension to raws-based mods.

A DFHack script is a Lua file that can be run as a command in DFHack. Scripts can do pretty much anything, from displaying information to enforcing new game mechanics. If you don't already know Lua, there's a great primer at lua.org.

6.1.2 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. However, DFHack scripts can do many things that you just can't do in the raws, like make a creature that trails smoke or launch a unit into the air when they are hit with a certain type of projectile. Some things *could* be done in the raws, but a script is better (e.g. easier to maintain, easier to extend, and/or 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 as an intermediary 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.

6.1.3 The structure of a mod

In the example below, we'll use a mod name of `example-mod`. I'm sure your mods will have more creative names! Mods have a basic structure that looks like this:

```
info.txt
graphics/...
objects/...
blueprints/...
scripts_modactive/example-mod.lua
scripts_modactive/internal/example-mod/...
scripts_modinstalled/...
README.md (optional)
```

Let's go through that line by line.

- **The `info.txt` file contains metadata about your mod that DF will**
  display in-game. You can read more about this file in the Official DF Modding Guide.
- **Modifications to the game raws (potentially with**
  custom raw tokens) go in the `graphics/` and `objects/` folders. You can read more about the files that go in these directories on the Modding wiki page.
- **Any *quickfort* blueprints included with your mod go in the**
  `blueprints` folder. Note that your mod can *just* be blueprints and the `info.txt` file if you like. See the next section for an example.
- **A control script in `scripts_modactive/` directory that handles**
  system-level event hooks (e.g. reloading state when a world is loaded), registering *overlays*, and *enabling/disabling* your mod. You can put other scripts in this directory as well if you want them to appear as runnable DFHack commands when your mod is active for the current world. Lua modules that your main scripts use, but which don't need to be directly runnable by the player, should go in a subdirectory under `scripts_modactive/internal/` so they don't show up in the DFHack *launcher* command autocomplete lists.
- **Scripts that you want to be available before a world is loaded (i.e. on the**
  DF title screen) or that you want to be runnable in any world, regardless of whether your mod is active, should go in the `scripts_modinstalled/` folder. You can also have an `internal/` subfolder in here for private modules if you like.
- **Finally, a `README.md` file that has more information about your mod.**
  If you develop your mod using version control (recommended!), that `README.md` file can also serve as your git repository documentation.

These files end up in a subdirectory under `mods/` when players copy them in or install them from the Steam Workshop, and in `data/installed_mods/` when the mod is selected as "active" for the first time.

6.1.4 What if I just want to distribute quickfort blueprints?

For this, all you need is `info.txt` and your blueprints.

Your `info.txt` could look something like this:

```
[ID:drooble_blueprints]
[NUMERIC_VERSION:1]
[DISPLAYED_VERSION:1.0.0]
[EARLIEST_COMPATIBLE_NUMERIC_VERSION:1]
[EARLIEST_COMPATIBLE_DISPLAYED_VERSION:1.0.0]
[AUTHOR:Drooble]
[NAME:Drooble's blueprints]
[DESCRIPTION:Useful quickfort blueprints for any occasion.]
[STEAM_TITLE:Drooble's blueprints]
[STEAM_DESCRIPTION:Useful quickfort blueprints for any occasion.]
[STEAM_TAG:dfhack]
[STEAM_TAG:quickfort]
[STEAM_TAG:blueprints]
```

and your blueprints, which could be .csv or .xlsx files, would go in a `blueprints/` subdirectory. If you add blueprint file named `blueprints/bedrooms.csv`, then it will be shown to players as `drooble_blueprints/bedrooms.csv` in *quickfort* and *gui/quickfort*. The "drooble_blueprints" prefix comes from the mod ID specified in `info.txt`.

6.1.5 What if I just want to distribute a simple script?

If your mod is just a script with no raws modifications, things get a bit simpler. All you need is:

```
info.txt
scripts_modinstalled/yourscript.lua
README.md (optional)
```

Adding your script to the `scripts_modinstalled/` folder will allow DFHack to find it and add your mod to the *Script paths*. Your script will be runnable from the title screen and in any loaded world, regardless of whether your mod is explicitly "active".

6.1.6 A mod-maker's development environment

Create a folder for development somewhere outside your Dwarf Fortress installation directory (e.g. `/path/to/mymods/`). If you work on multiple mods, you might want to make a subdirectory for each mod.

If you have changes to the raws, you'll have to copy them into DF's `data/installed_mods/` folder to have them take effect, but you can set things up so that scripts are run directly from your dev directory. This way, you can edit your scripts and have the changes available in the game immediately: no copying, no restarting.

How does this magic work? Just add a line like this to your `dfhack-config/script-paths.txt` file:

```
+/path/to/mymods/example-mod/scripts_modinstalled
```

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 other versions of your mod in the DF mod folders).

6.1.7 The structure of the game

"The game" is in the global variable *df*. Most of the information relevant to a script is found in `df.global.world`, which contains things like the list of all items, whether to reindex pathfinding, et cetera. Also relevant to us are the various data 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.

6.1.8 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, `unit` will be `nil` and an error message will be printed (which can be silenced by passing `true` to `getSelectedUnit`).

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, 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 from *gui/launcher* when a unit is selected in the Dwarf Fortress UI.

6.1.9 Exploring DF state

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 layouts of the game's structures. These are complete, but difficult to read (for a human). The second option is the *gui/gm-editor* interface, an interactive data explorer. You can run the script while objects like units are selected to view the data within them. 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*.

6.1.10 Reacting to events

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 callbacks too 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:

```
local repeatUtil = require('repeat-util')
```

Both `repeat-util` and `eventful` require keys for registered callbacks. You should use something unique, like your mod name:

```
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:

```
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:

```
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:

```
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.

6.1.11 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]
```

(continues on next page)

(continued from previous page)

```
    [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:

```
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
```

(continues on next page)

(continued from previous page)

```
        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
```

(continues on next page)

(continued from previous page)

```
        dfhack.units.subtractActionTimers(unit, amount,
                df.unit_action_type_group.MovementFeet)
    end
end
```

6.1.12 Putting it all together

Ok, you're all set up! Now, let's take a look at an example `scripts_modinstalled/example-mod.lua` file:

```
-- main file for example-mod

-- these lines indicate 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"
--@module = true
--@enable = true

-- this is the help text that will appear in `help` and
-- `gui/launcher`. see possible tags here:
-- https://docs.dfhack.org/en/latest/docs/Tags.html
--[====[
example-mod
===========

Tags: fort | gameplay

Short one-sentence description ...

Longer description ...

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('internal/example-mod/module-a')
local moduleB = reqscript('internal/example-mod/module-b')
local moduleC = reqscript('internal/example-mod/module-c')
local moduleD = reqscript('internal/example-mod/module-d')

local GLOBAL_KEY = 'example-mod'

enabled = enabled or false
```

(continues on next page)

(continued from previous page)

```
function isEnabled()
    -- this function is for the enabled API, the script won't show up on the
    -- control panel without it
    return enabled
end

dfhack.onStateChange[GLOBAL_KEY] = function(sc)
    if sc == SC_MAP_UNLOADED then
        dfhack.run_command('disable', 'example-mod')

        -- ensure our mod doesn't try to enable itself when a different
        -- world is loaded where we are *not* active
        dfhack.onStateChange[GLOBAL_KEY] = nil

        return
    end

    if sc ~= SC_MAP_LOADED or df.global.gamemode ~= df.game_mode.DWARF then
        return
    end

    dfhack.run_command('enable', 'example-mod')
end

if dfhack_flags.module then
    return
end

if not dfhack_flags.enable then
    print(dfhack.script_help())
    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)
```

(continues on next page)

(continued from previous page)

```
    -- 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')
    enabled = false
end
```

Inside `scripts_modinstalled/internal/example-mod/module-a.lua` you could have code like this:

```
--@ module = true

function onLoad() -- global variables are exported
    -- do initialization here
end

-- this is a local function: local functions/variables
-- are not accessible to other scripts.
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!

6.2 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.

6.2.1 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 (werecreature-ready), library, temple, 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 walkthrough by running *gui/quickfort*, searching for `dreamfort help`, and selecting the blueprints.

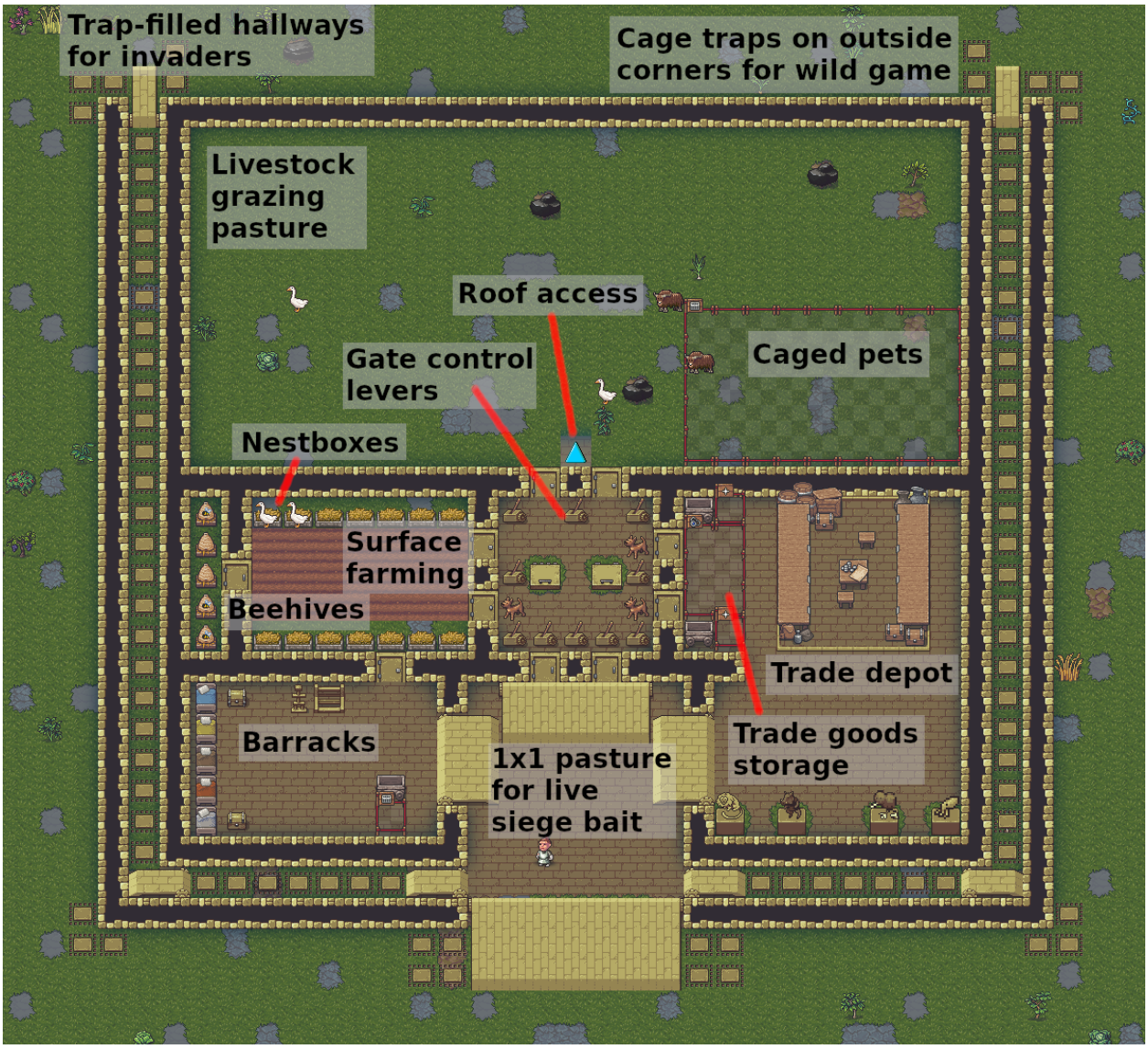
Dreamfort blueprints are available for easy viewing and copying online.

The online spreadsheets also include embark profile suggestions, a complete example embark profile, and a convenient checklist that you can use to track your progress.

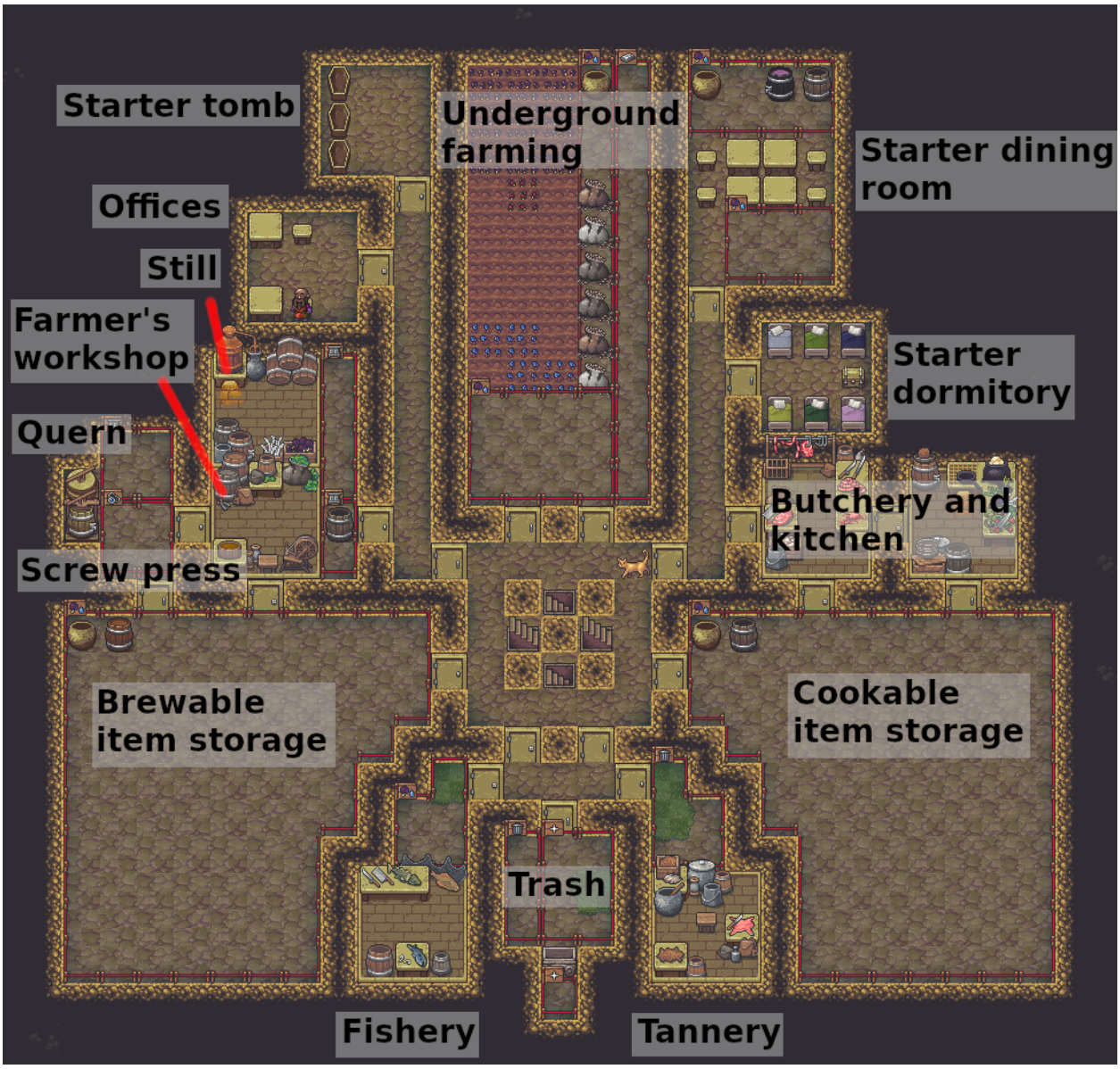
If you like, you can download a fully built Dreamfort-based fort from dffd, load it, and explore it interactively.

Here are annotated screenshots of the major Dreamfort levels (or click here for a slideshow).

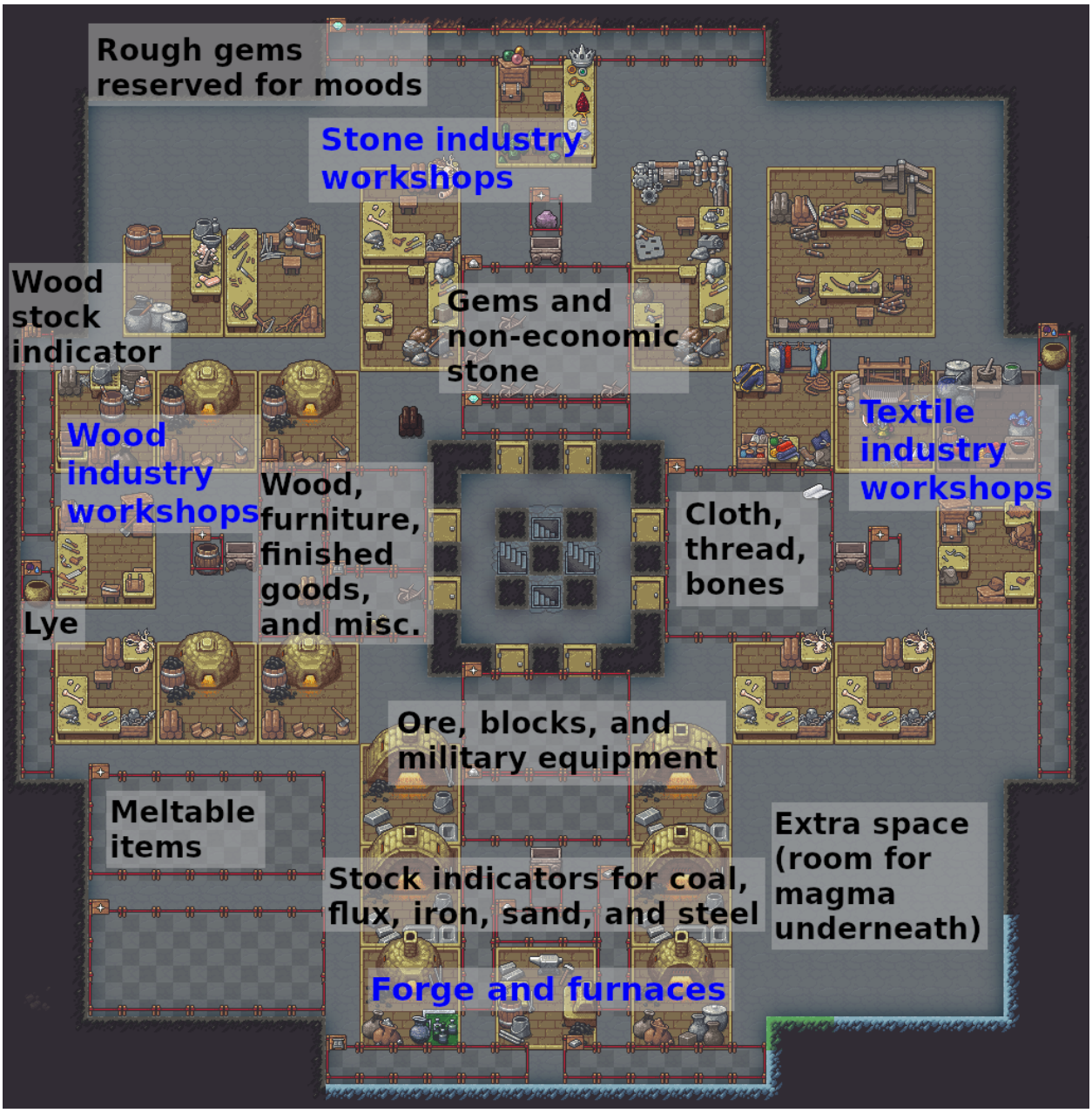
Surface level



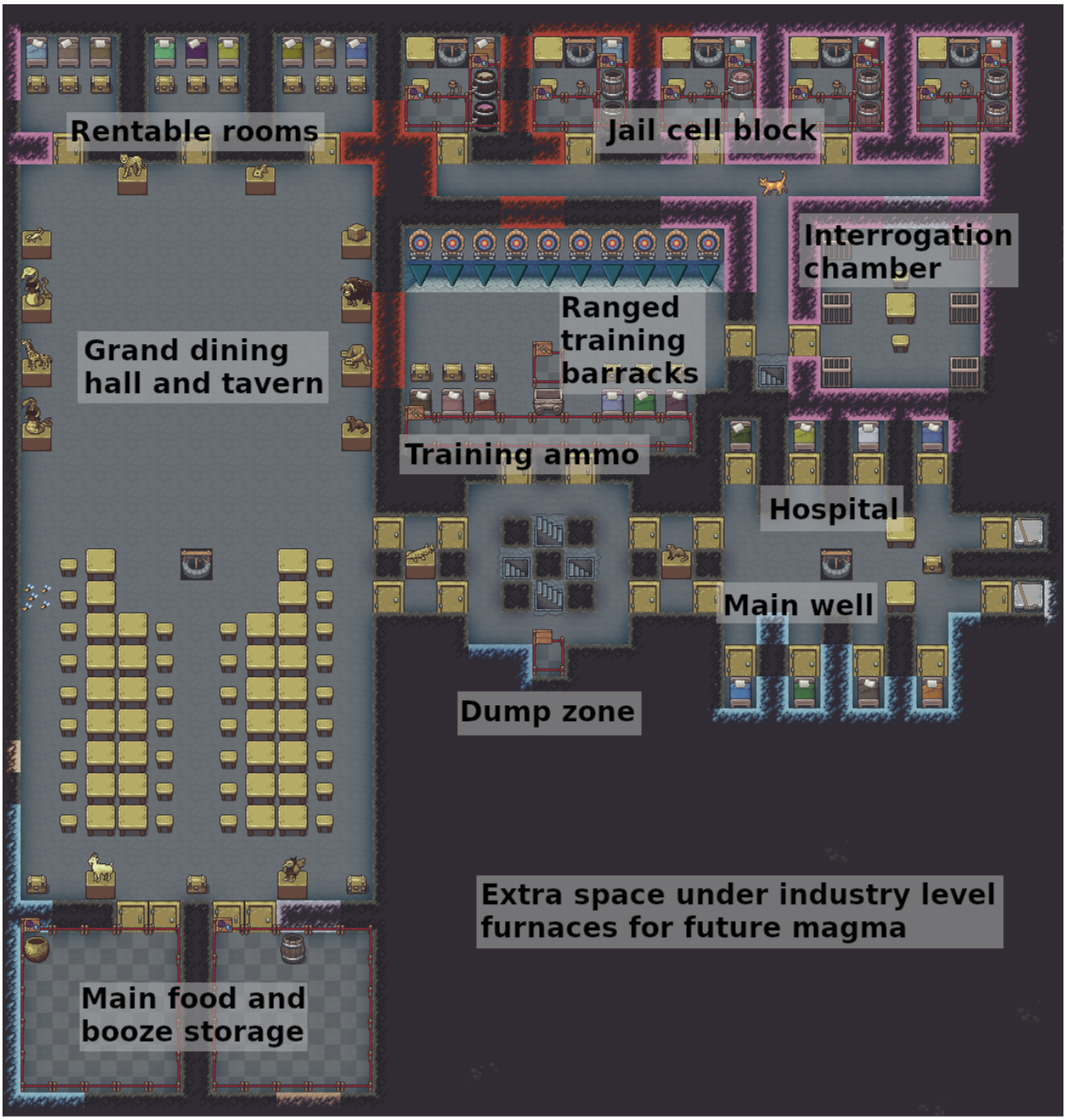
Farming level



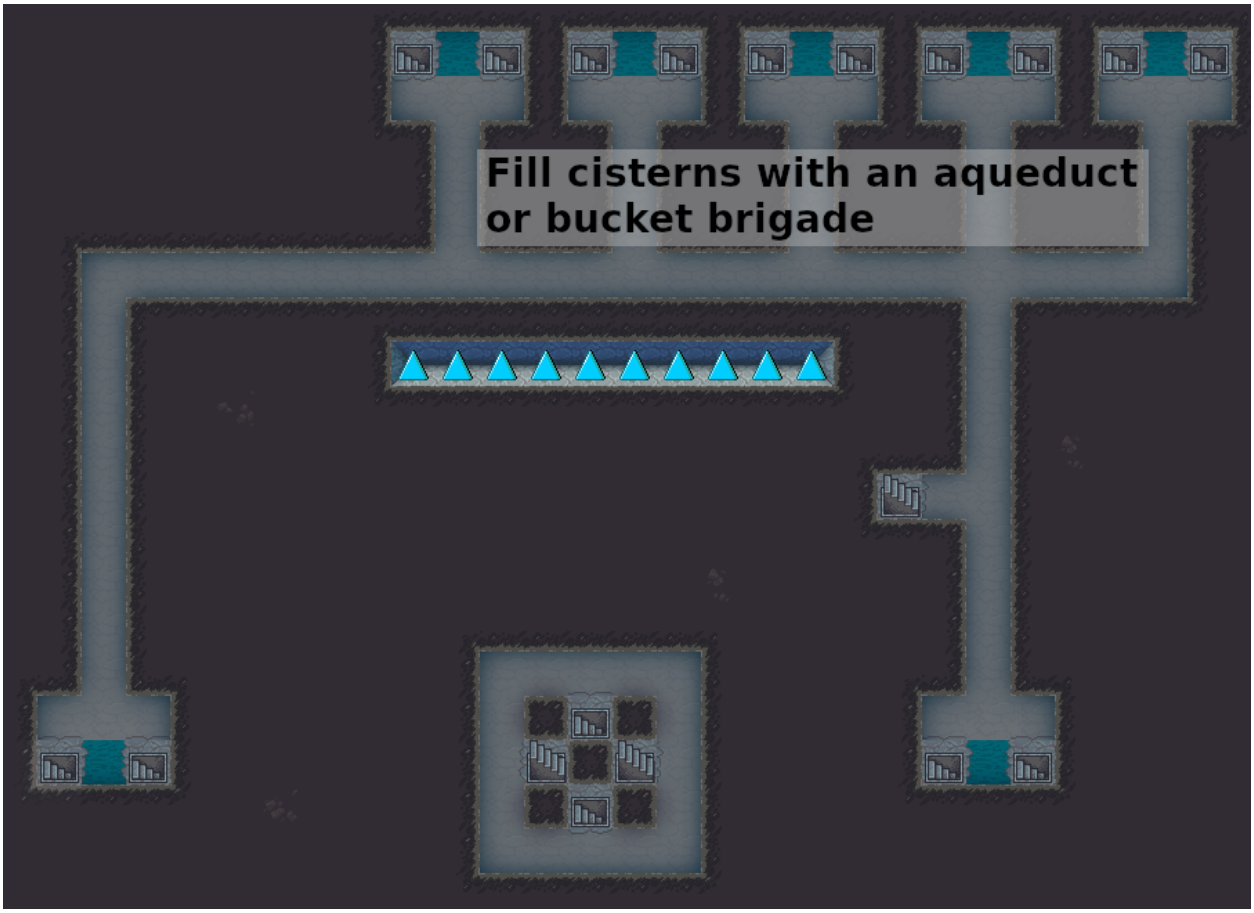
Industry level



Services levels (4 deep)

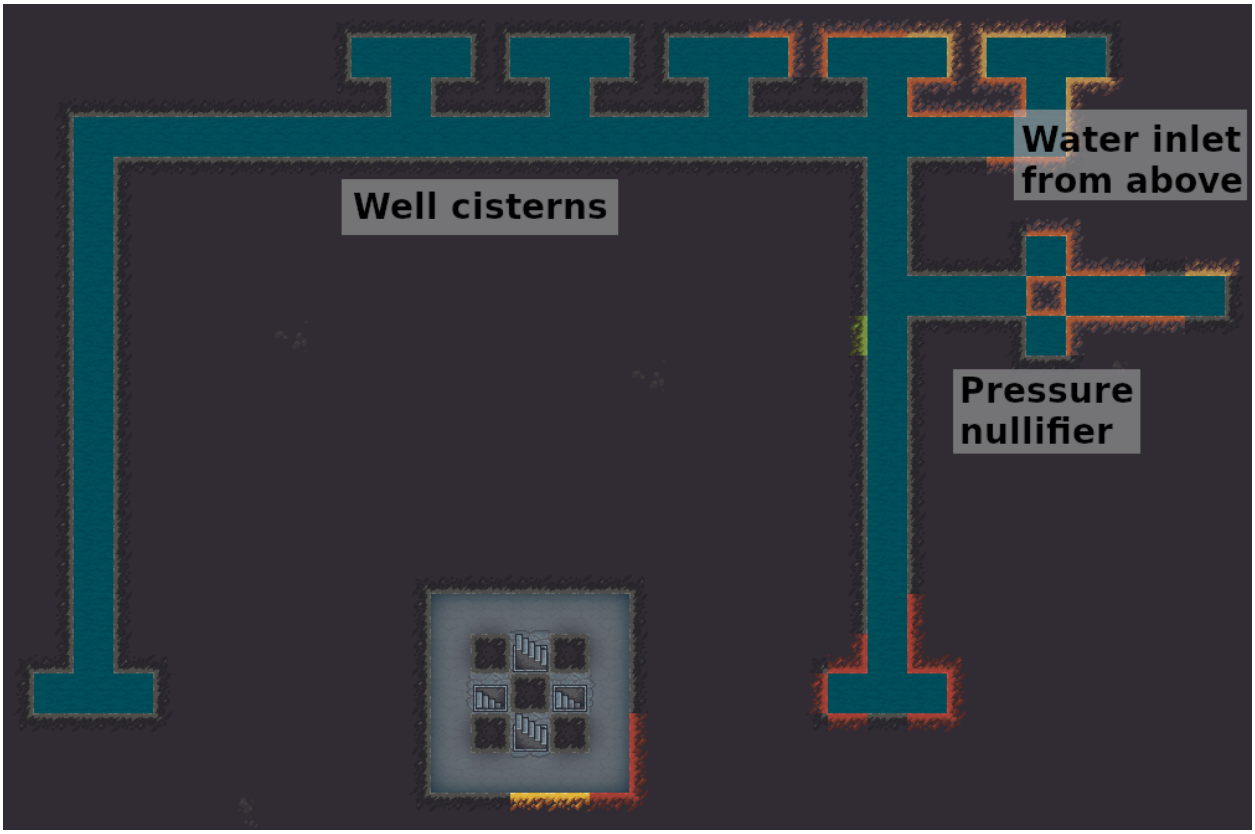


Services waterway:



Example plumbing to fill cisterns

If you are routing water to fill the cisterns, you can do it like this (1 z-level below the preceding screenshot)



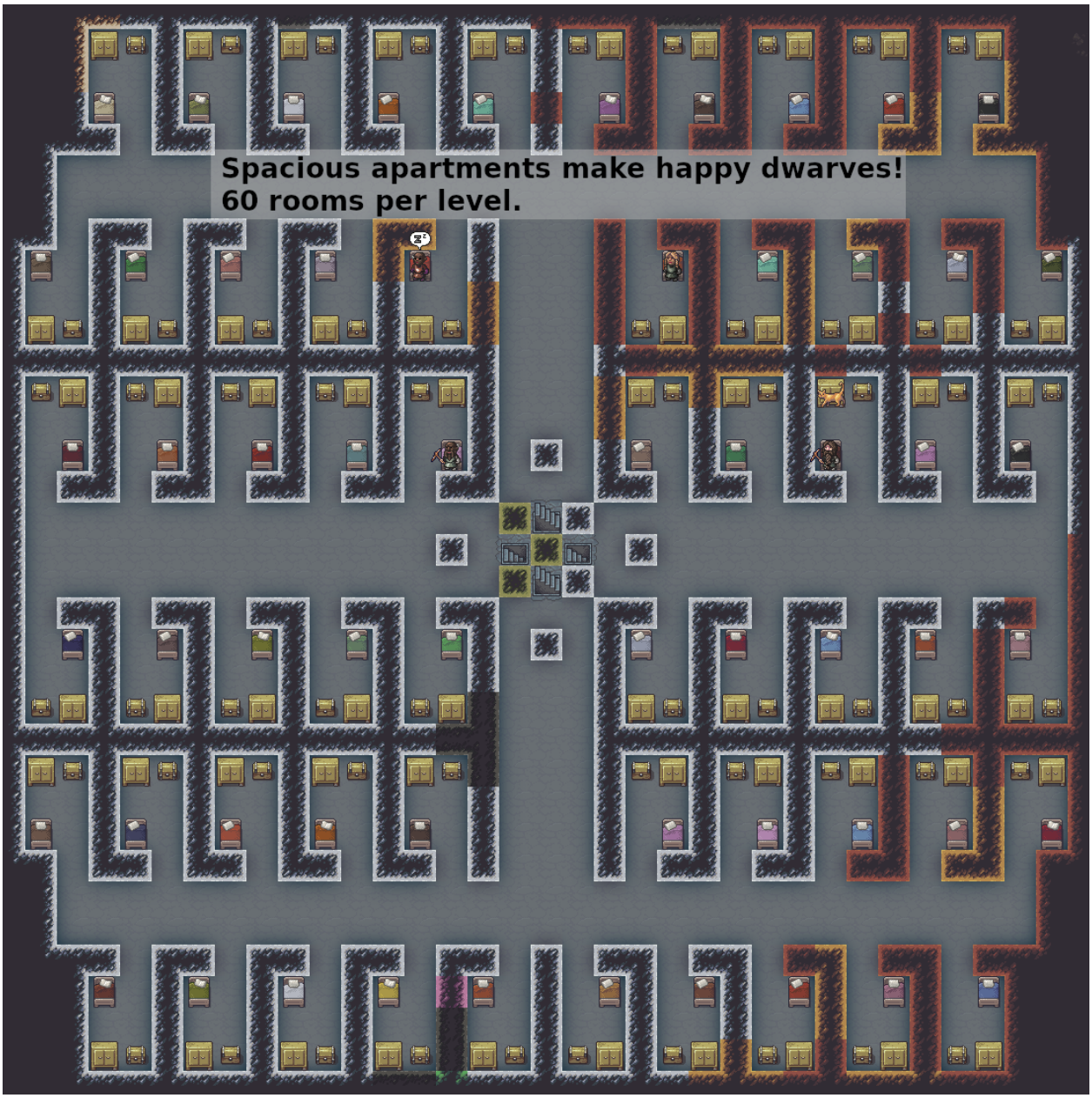
Cistern drain (keep open while you're digging out the aquifer tap):


Be sure to build an
emergency drain!

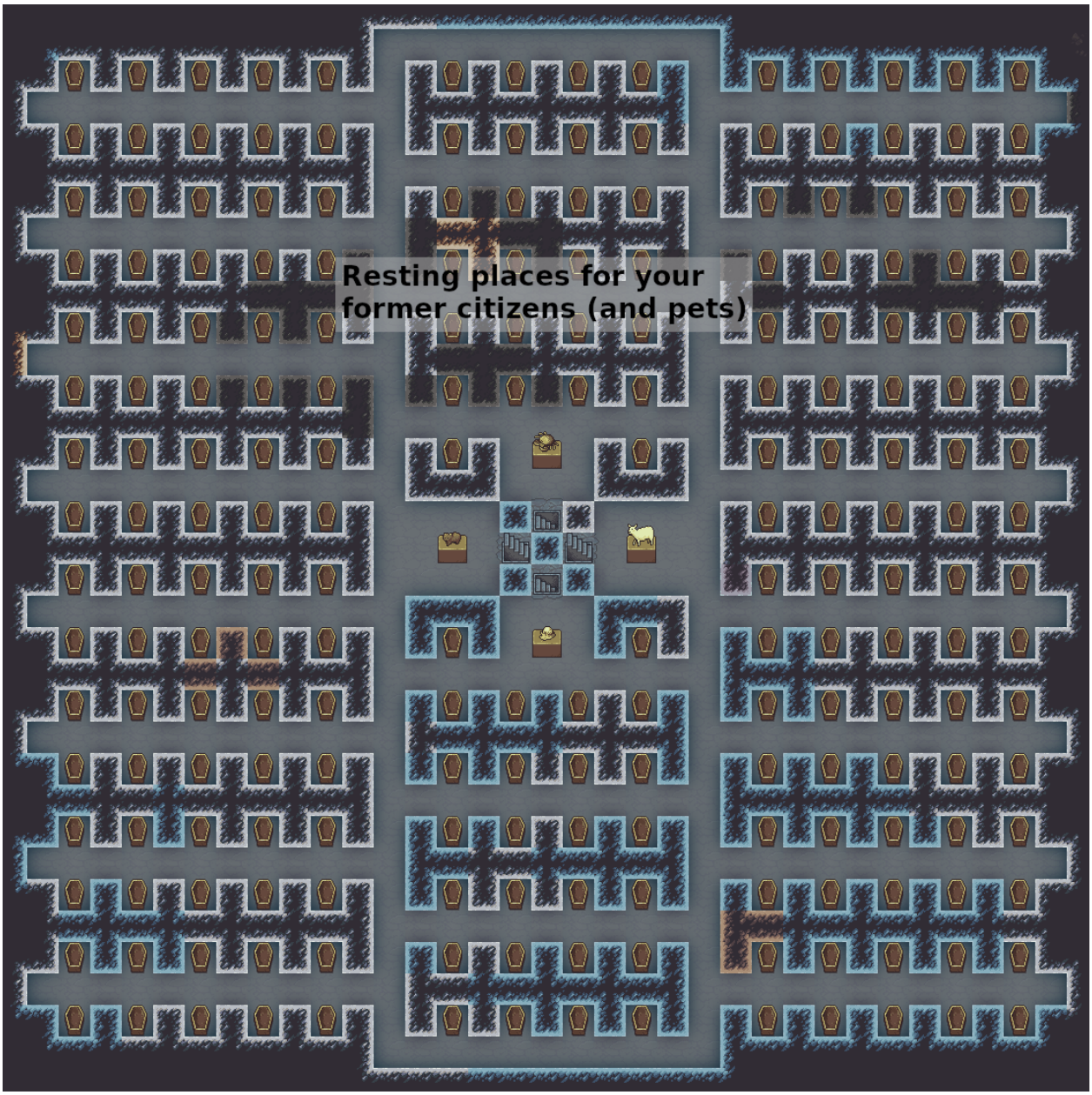
Guildhall level

Noble suites

Apartments



Crypt



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.

6.2.2 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

6.2.3 Bedrooms

These are popular bedroom layouts from the Bedroom design page on the wiki. Each file has blueprints to dig the rooms, zone them as bedrooms, and build the furniture.

- 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

6.2.4 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

6.2.5 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 *gui/quickfort*'s `repeat` functionality) for exploring through the depths.

- library/exploratory-mining/tunnels.csv
- library/exploratory-mining/vertical-mineshafts.csv
- library/exploratory-mining/connected-mineshafts.csv

6.2.6 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 the `library/aquifer_tap.csv /help` blueprint. Alternately, view the demo video below.

You can see how to nullify the water pressure (so you don't flood your fort) in the Dreamfort cistern screenshot above: *Services levels (4 deep)*.

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 the `library/pump_stack.csv /help` blueprint or by viewing the demo video:

The blueprint spreadsheet is also available online.

6.3 Quickfort blueprint creation guide

Quickfort is a DFHack tool 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 actions are supported through the use of multiple files or spreadsheets, each describing a different phase of DF construction: designating digging, defining zones, placing stockpiles, and building.

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, preserving compatibility with existing blueprints (where possible – DF itself has changed since then). In contrast with the earlier quickfort implementations, which interacted with DF by simulating keyboard input, DFHack Quickfort calls lower-level API functions to designate tiles and configure buildings. This allows for nearly instantaneous blueprint application, error checking and recovery, and many other advanced features.

This guide focuses on DFHack Quickfort's capabilities and blueprint syntax, 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.

If you are just looking to apply existing blueprints to your fort, check out *gui/quickfort* (or *quickfort* for the commandline version). There are many ready-to-use blueprints available in the *Quickfort blueprint library* distributed with DFHack.

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 that section of your map using *gui/blueprint*. 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

6.3.1 Feature summary

- General
    - Blueprint modes for all phases of fort design
    - Read blueprints from .csv or multi-worksheet .xlsx files
    - Near-instant application, even for very large and complex blueprints
    - Blueprints can span multiple z-levels
    - Easy sharing of blueprints with multi-blueprint files
    - Scripted application of sequences of blueprints
    - Undo applied blueprints

  - Rotate blueprints or flip them around
  - Automatic cropping of blueprints that extend off the map
  - Generate manager orders for items required by a blueprint
  - Includes a library of ready-to-use blueprints
  - Blueprint debugging
- 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
- Zone and place modes
  - Define zones and stockpiles of any shape, not just rectangles
  - Configurable numbers of bins, barrels and wheelbarrows assigned to created stockpiles
  - Automatic splitting of stockpiles that exceed maximum dimension limits
  - Create and attach locations to zones
  - Full control over stockpile configuration based on the *stockpiles* settings library
  - Configurable zone/location settings, such as the pit/pond toggle or hospital supply quantities
- Build mode
  - Integrated with DFHack *buildingplan*: you can place buildings before manufacturing building materials and you can use the *buildingplan* UI for setting materials and quality 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 causing errors and interrupting 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)
  - Set building properties (such as a name)
  - Can attach and configure track stops as part of hauling routes
- Burrow mode
  - Supports creating, adding to, and subtracting from burrows.

6.3.2 Introduction to 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 specifying designations. The following "mode" keywords are understood:

Blueprint mode	Description
dig	Designations (digging, traffic, dumping, etc.)
build	Constructions, buildings, and furniture
place	Stockpiles
zone	Activity zones

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 design operations, 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` 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 #
# # # # #
```

The letter `d` here stands for "dig". The character sequences in these blueprints are based on the old (pre-v50) keyboard shortcuts for the various DF menus. Please see the *Appendix: Symbols and properties* below for a full reference.

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.

In general, any cell that contains text that starts with a `#` is interpreted as a comment and is ignored by *quickfort*. You can use this to leave notes for yourself inside of a blueprint. Take care to start your comment with a space after the `#` to avoid accidentally starting a modeline if your comment happens to be in the first column and happens to start with a modeline keyword. For example, `#dig this area out` is an accidental modeline that will cause problems. However, `# dig this area out` is a safe comment.

Once the dwarves have that dug out, let's zone it as a bedroom:

```
#zone
b b b b #
b b b b #
b b b b #
b b b b #
# # # # #
```

This looks very similar to the `#dig` blueprint above, but with `b`s instead of `d`s. The `b`s mark the area for a bedroom zone just like the `#dig` blueprint marked the area for digging. It's important to wait until after the area is completely dug out before applying further blueprints since zones can't be applied to hidden tiles and furniture can't be built in undug walls.

Now, let's add some walls and furniture:

```
#build
Cw Cw Cw Cw #
Cw b  h  Cw #
Cw       Cw #
Cw Cw    Cw #
#  #  #  #  #
```

The `Cw` cells represent the constructed walls, leaving space for a door that we might want to add later. Quickfort uses *buildingplan* for managing buildings, so the walls will be built out of whatever matches the current buildingplan filter set for walls. Also note my generosity – in addition to the bed (`b`) I've built a container (`h`) for this lucky dwarf.

Finally, let's place a booze stockpile in the 2 unoccupied tiles in the room:

```
#place personal booze stockpile
` ` ` ` #
` ~ ~ ` #
` f f{name="bedroom booze"}:=booze
` `   ` #
# # # # #
```

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. You can also treat disconnected segments as belonging to the same stockpile, but we'll get into *Label syntax* later.

Now what's all that business attached to the second `f`? The part between the curly brackets specifies properties, in this case the name that we want to give the stockpile. The remaining part, from the colon (`:`) onward, applies the `booze` preset from the *stockpiles* library. That will configure the stockpile to accept only booze. You can use presets (along with other options that we'll go over later) to configure stockpiles however you want, directly from the `#place` blueprint.

And that's it! You now have a series of blueprints that you can "stamp" across your fort to quickly build new bedrooms.

Area expansion syntax

In Quickfort, the following blueprints are equivalent:

```
#dig a 3x3 area
d d d #
d d d #
d d d #
# # # #

#dig the same area with d(3x3) specified in row 1, col 1
d(3x3)#
` ` ` #
` ` ` #
# # # #
```

The second example uses Quickfort's "area expansion syntax", which takes the form:

```
text(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 #
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) `  #
#  #  #  #  #
```

Property syntax

Many things you can designate with *quickfort* are configurable. All buildings, stockpiles, and zones, for example, can be named. These configuration elements are expressed as properties.

Properties are written between curly brackets ({}). There can be multiple properties defined between those brackets, separated by spaces. Each property has a name and a value, with an equal sign to connect them. If a property value has a space within it, it should be surrounded by double quotes (").

If you have defined the area of something over multiple spreadsheet cells, you can specify properties in just one of those cells and they will apply to the whole object. You can even split properties up among multiple cells if that is more convenient. If you are using expansion syntax, the expansion part always goes last.

Here's an example of a seed stockpile that is configured to take from a seed feeder stockpile:

```
#place
f{name=Seeds links_only=true}:=seeds(3x2)

f
f{name="Seeds feeder" give_to=Seeds}:=seeds
f{containers=0}
```

Different modes and different types may have different properties that you can configure. See the *Appendix: Symbols and properties* for a full list.

Label syntax

Labels are different from the `name` property. They are only used internally by Quickfort to associate tiles with a particular zones or stockpiles. This is useful for when you want to define two touching zones or stockpiles of the same type(s), but you can't use expansion syntax because they are non-rectangular. It is also useful for marking two *disconnected* regions as belonging to the same zone or stockpile. Note that every tile in the zone or stockpile must be marked with the same label:

```
#place two touching stockpiles of the same type
f/feed f/feed f/feed{name="Seeds feeder" containers=0}:=seeds
f/feed f      f/feed
f      f      f{name=Seeds links_only=true take_from="Seeds feeder"}:=seeds

#zone one pasture in two disconnected regions
n/slots n/slots n/slots

n/slots{name="Pasture slots"}(3x1)
```

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 stonecutter 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 stonecutter 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 stonecutter 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 stonecutter workshops all in a row:

```
#build line of 20 stonecutter 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)
```

6.3.3 #dig mode

`#dig` blueprints are normally the first step in any design. They define the boundaries and layouts for the blueprints for later stages of construction. Despite their name, `#dig` blueprints are for more than just digging. They also handle smoothing, carving, traffic designations, and marking items on the ground for dumping, forbidding, or other similar tags. See the full list of supported designations in the *#dig mode reference*.

Dig priorities

DF designation priorities are supported in `#dig` blueprints. The full syntax is `[markers][symbol][number][expansion]`, where if the `symbol` is not specified, `d` is assumed, and if `number` is not specified, `4` is assumed (the default priority). So all of these blueprints are 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  #
#  #  #  #  #  #
```

(continues on next page)

(continued from previous page)

```
#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  #
#  #  #  #  #  #
```

At least one of the symbol and the priority number must be specified.

Dig markers

There are three types of markers you can apply to dig designated tiles. You can apply multiple markers to the same tile by including multiple marker prefixes. For example, a tile that is in blueprint mode and that is also marked for warm and damp dig would be written as:

```
#dig
mbmwmdd
```

Blueprint marker

"Blueprint" markers are useful for when you want to plan out your digging, but you don't want to dig everything just yet. Here, "blueprint" refers to the vanilla UI "blueprint mode" vs. "standard mode" buttons. You can use DF's "Change blueprints to standard selections" button to convert the "blueprint" marked tiles to regular designations.

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 mbd mbd mbd d #
d mbd mbd mbd d #
d mbd mbd mbd d #
d   d   d   d d #
#   #   #   # # #
```

To apply an entire dig blueprint in blueprint 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 by running `quickfort set force_marker_mode true`.

Note that the state of the in-game vanilla button that you use to draw designations in either Standard or "Blueprint" mode does not have any effect on *quickfort*.

Warm and damp dig markers

Warm and damp dig markers allow digging to continue uninterrupted through warm or damp tiles. These markers are useful to include in blueprints that are expected to be applied near magma or in damp/aquifer layers. See the *dig* tool for more info on warm and damp dig.

The prefix for warm dig is `mw` and the prefix for damp dig is `md`.

Carved minecart 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 `track` 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) #
`      ` `      #
```

(continues on next page)

(continued from previous page)

```
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.

6.3.4 #zone mode

Zones define how regions of your fort should be treated. They are also the anchor point for "locations" like taverns and hospitals. Unlike stockpiles or buildings, zones can overlap, which can lead to some interesting layouts. See the full list of zone symbols in the *#zone mode reference*.

Zone designation syntax

A zone is declared with a symbol followed by optional properties:

```
#zone a single tile garbage dump zone
d

#zone a single tile garbage dump zone named "The Dump"
d{name="The Dump"}
```

(continues on next page)

(continued from previous page)

```
#zone interrogation room
o{name=Interrogation assigned_unit=sheriff}

#zone a small inactive pond zone
p{name="Fill me" pond=true active=false}(3x3)
```

If you want multiple zones that have the same footprint, they can be declared from the same cell:

```
#zone pasture and training area
n{name="Main pasture"}t{name="Pet training area"}(14x10)
```

or from different corners of the same rectangle:

```
#zone pasture and training area
n{name="Main pasture"}(10x2)
t{name="Pet training area"}(10x-2)
```

and you can use this technique to achieve partial overlap, of course. The only configuration that can't be specified in a single blueprint is multiple non-rectangular zones that are partially overlapping. You will have to use multiple `#zone` blueprints to achieve that.

You can also use labels (see *Label syntax* above) to separate adjacent non-rectangular zones that happen to be of the same type or to combine disconnected regions into a single zone.

Locations, locations, locations

Hospitals, guildhalls, taverns, libraries, and temples are locations. You can declare a location in the properties for a zone:

```
#zone metalcrafter hall
m{location=guildhall profession=metalcrafter}(7x7)
```

You can attach multiple zones to a single location by giving the location a label (not a name – you can name zones, but you can't directly name locations) and then using that label for each of the zones you want to attach:

```
#zone tavern and rented room
b{location=tavern/bigpub name="Rent me"}(3x1)
h{location=tavern/bigpub name="Central pub" allow=residents}(25x40)
```

Note that the label ("bigpub" in this case) will never appear in-game. It is only used in the context of the blueprint to identify a common location.

6.3.5 #place mode

`#place` mode is dedicated to stockpiles, which are a major design element in any fortress.

Stockpile designation syntax

Just like zones, stockpiles can have properties like names or lists of other stockpiles to take from. Unlike zones, stockpiles can have configuration specifiers for exactly what types of items to accept. The full syntax looks like this:

```
types/label{properties}:configuration(expansion)
```

with every component other than the type being optional. You're already familiar with *Property syntax*, *Label syntax*, and *Area expansion syntax*, so let's focus in on the remaining elements.

Stockpile types

The type of stockpile corresponds to the category of items it accepts. Some types will cause the stockpile to accept bins or barrels. See the full list in the *#place mode reference*.

It is very common to have stockpiles that accept multiple categories of items. Although it is perfectly valid to declare a single-purpose stockpile, *quickfort* also supports directly declaring all the categories at once. For example, to declare a 20x10 stockpile that accepts both corpses and refuse, you could write:

```
#place refuse heap
yr(20x10)
```

The order of the individual letters doesn't matter. If you want to configure the stockpile from scratch, you can place unconfigured "custom" stockpiles with (c).

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 in the properties.

Individual properties for `bins`, `barrels`, and `wheelbarrows` are supported. You can also set them all at once with the `containers` alias (it usually just makes sense to set this to 0 when you don't want any containers of any type). For example:

```
#place a stone stockpile with five wheelbarrows
s{wheelbarrows=5}(3x3)

#place a bar, ammo, weapon, and armor stockpile with 20 bins
bzpd{bins=20}(5x5)

#place a weapon stockpile with no bins
p{containers=0}(9x2)
```

That last one could have equivalently used `bins=0`, but sometimes you just don't want to have to think about which stockpile types take which type of container.

The container settings also have a shorthand form. If you add a number after a type symbol, you can set the relevant container count. The first example above could equivalently be written as:

```
#place a stone stockpile with five wheelbarrows
s5(3x3)
```

It sets the count for wheelbarrows specifically because the container associated with stone stockpiles is wheelbarrows.

If a stockpile has multiple types, it is the just the previous type symbol that matters. `s5e` (stone and gems) would still set wheelbarrows only since the `5` comes directly after the `s`. `se5` would set bins and not wheelbarrows since the `5` would affect the container type associated with gem stockpiles: bins.

If the number follows a type symbol that does not have a specific container type associated with it, then the container type defaults to wheelbarrows. This allows you to easily add wheelbarrows to furniture and corpse stockpiles, where having wheelbarrows is useful, but not added by default by the game.

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!

Generating manager orders for 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, the following blueprint will enqueue orders for 10 rock pots, even though the stockpile only has 9 tiles:

```
#place
f{barrels=10}(3x3)
```

Stockpile configuration

Quickfort uses the *stockpiles* plugin and *The stockpiles settings library* to configure stockpile settings, and provides a syntax that is easy to write in a blueprint yet still allows you to access the full power of the *stockpiles* command.

The syntax is:

```
: <op> <preset_name> [/<filter>] [<op> <preset_name> [/<filter>]...]
```

<op> is one of =, -, or +, representing the three *stockpiles* import modes: `set`, `disable`, or `enable`, respectively. Note that if you are using an = op, then it should go first in the list. Any = configuration segment will override anything that comes before it.

For example, a blueprint like:

```
#place
f:=booze(5x4)
```

would be equivalent to creating a 5x4 food stockpile in the UI, then selecting it and running this command:

```
stockpiles import --mode=set booze
```

you can also add a slash (/) and a comma-separated list of filter strings to customize the settings further:

```
#place
p{name="Metal weapons"}:-cat_weapons/other/(7x3)
```

Note that the "op" in this case lets us disable the matched preset, which in this case is the "Other materials" types in the Weapons category. This configuration is equivalent to the *stockpiles* command:

```
stockpiles import --mode=disable cat_weapons --filter=other/
```

And we can chain multiple *stockpiles* commands together by adding another "op" character and another preset:

```
#place
p{name="Steel weapons"}:-cat_weapons/mats/,other/+steelweapons(7x3)
```

which corresponds to running these two commands:

```
stockpiles import --mode=disable cat_weapons --filter=mats/,other/
stockpiles import --mode=enable steelweapons
```

With the combination of the library presets and custom filter strings, you can configure any stockpile any way you like!

6.3.6 #build mode

`#build` mode handles buildings, furniture (which are also "buildings" according to DF), constructions (including constructed tracks), and hauling routes.

Building designation syntax

The syntax is very similar to the syntax for stockpiles, except that it only makes sense to have a single symbol to indicate what to build on that tile:

```
symbol{properties}:configuration(expansion)
```

See the *#build mode reference* for properties that you can specify for each building type.

Here's an example of a simple 5x5 square of flooring:

```
#build
Cf(5x5)
```

or a named Jeweler's workshop that takes from specific stockpiles:

```
#build
wj{name="Encrusting center" take_from="Furniture,Gem storage"}
```

or a forge that specializes in high-quality armor:

```
#build
wf{name=Armorer labors=Armoring min_skill=Master}
```

The `:configuration` part is only relevant for hauling routes, which we'll go over in the next section.

Hauling route definitions

Hauling routes are defined by properties and configuration attached to track stops. You can define a single-stop hauling route for a quantum stockpile as easily as a multi-stop stone quarry transportation line. The stockpile-like `:configuration` part of the syntax controls which item types are considered "desired" for the hauling route stop. If it's not specified, then all item types are accepted. This is the most common case since most hauling route contents are filtered by the stockpiles that the stops take from, but the flexibility is there for when multiple stops take different items from the same stockpile, or when a stop only wants a subset of items from a stockpile.

Here is a common setup for a quantum stone stockpile:

```
#place
s{name="Stone quantum" quantum=true} ~ s5{name="Stone feeder"}(3x3)
#build
~ trackstopW{take_from="Stone feeder" route="Stone dumper"}
```

This sets up the quantum stockpile and the feeder stockpile in the `#place` blueprint, followed by the trackstop and the hauling route configuration in the `#build` blueprint. The `route` property is the name of the hauling route to create (or attach to if it already exists). If you are applying a quantum stockpile blueprint more than once in a fort, be sure to *avoid* defining the `route` property so that each application of the blueprint creates a unique hauling route. Two quantum stockpiles on the same route will not function properly (since one of the stops will be missing a minecart).

Let's look at a slightly more complicated setup where we sort the stone into different output quantum stockpiles:

```
#place
s{name="Other stone quantum" quantum=true}    ~ s5e{name="Rock feeder"}(3x3)
s{name="Ore/clay stone quantum" quantum=true} ~
s{name="Gem quantum" quantum=true}            ~
#build
~ trackstopW{take_from="Rock feeder" route="Other stone"}:=otherstone
~ trackstopW{take_from="Rock feeder" route="Ore/clay"}:=cat_stone-otherstone
~ trackstopW{take_from="Rock feeder" route="Gems"}:=cat_gems
```

You can see how we make use of the stockpile-style configuration syntax to fine-tune the items desired by the hauling route stop.

Finally, let's make a series of stops on a common hauling route. There is nothing particularly special about this example. If the `route` property names an existing route, the stop will be added to that route:

```
#dig
trackE trackEW trackEW trackW
#build
trackstop{route="Tick tock"} ~ ~ trackstop{route="Tick tock"}
```

These two track stops (which do not dump their contents) simply exist on a common route at the ends of a connected carved track.

6.3.7 #burrow mode

`#burrow` mode can create, extend, and remove tiles from burrows.

Burrow designation syntax

The syntax should look familiar by now:

```
symbol{properties}(expansion)
```

See the *#burrow mode reference* for symbol and property definitions.

Here's how to create (or add to, if a burrow by that name already exists) a 5x5 burrow named `Inside+`. It will also register this burrow with *gui/civ-alert* if no burrow has yet been registered:

```
#burrow
a{create=true name=Inside+ civalert=true}(5x5)
```

Why the trailing +? That's to indicate to the *burrow* plugin that the burrow should grow as adjacent tiles are dug out.

Similarly, here is how to erase a tile from all burrows that currently include it:

```
#burrow
e
```

6.3.8 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) No explicit 'start()' so cursor is in upper left
#meta label(digwholefort) start(center of stairs on surface)
#dig label(dig_dining) hidden() called by the digwholefort meta blueprint
#zone label(pastures) message(remember to assign animals to the pastures)
```

Blueprint labels

Labels are displayed in the blueprint selection dialog and the output of `quickfort list` 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(stonew) a stonecutter 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 blueprint listings 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 `stonew` blueprint above could look like this:

```
#meta start(center of workshop) a stonecutter workshop
/stonew
```

You can use semicolons, commas, or spaces to separate the elements of the `start()` marker, whichever you prefer.

Hiding blueprints

A blueprint with a `hidden()` marker won't appear in the blueprint listings unless hidden blueprints are specifically requested. 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. 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.

6.3.9 Other blueprint modes

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
notes	Display long messages, such as help text or blueprint walkthroughs
ignore	Hide a section of your spreadsheet from quickfort, useful for scratch space or personal notes
meta	Script sequences of blueprints together, transform them, and/or repeat them across multiple z-levels

#notes mode

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.
```

(continues on next page)

(continued from previous page)

```
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.

#ignore mode

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.

#meta mode

`#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.

Scripting blueprints together

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 another dig blueprint** to designate traffic costs
4. **Apply a zone blueprint** to designate zones
5. **Apply a place buildprint** to designate and configure stockpiles
6. **Apply a build buildprint** to designate buildings

Those last four "apply"s might as well get done in one command instead of four. 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) 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
...
#zone label(bed2) declare bedroom zones
```

(continues on next page)

(continued from previous page)

```
...
#place label(bed3) add food stockpiles
...
#build label(bed4) build the furniture
...
```

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 *gui/quickfort* or 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 last three blueprints into one:

```
"#meta label(bed234) combines zone, place, and build 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 declare zones, place stockpiles, and build furniture

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 label(dig_it) dig the whole fortress
dig_farming/1
#>
dig_industry/1
#>
dig_dining/main
#>
dig_dining/basement
#>
dig_dining/waterway
#>
```

(continues on next page)

(continued from previous page)

```
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.

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 the blueprint selection lists by adding a `hidden()` marker to their modelines. That way, the blueprint lists 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 by toggling the "Hidden" setting in the *gui/quickfort* load dialog or 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.

6.3.10 Packaging a set of blueprints

A complete specification for a section of your fortress may contain 4 or more separate blueprints, one for each "phase" of construction (dig, zone, place, build).

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 #
```

(continues on next page)

(continued from previous page)

```
# # # # #
#zone label(bed2)
b(4x4)  #
        #
        #
        #
# # # # #
#place label(bed3)
        #
f(2x2)  #
        #
       #
 # # # # #
#build label(bed4)
b   f h #
        #
        #
t c     #
# # # # #
```

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.zone.csv
bedroom.3.place.csv
bedroom.4.build.csv
```

The naming and organization is completely up to you.

6.3.11 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. *quickfort* uses *buildingplan* to manage construction.

This means that *buildingplan's filters* will be used 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. You can set the filters the way you want for one blueprint, run the blueprint, and then freely change the filters 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 can autogenerate manager orders for everything that a blueprint requires. See *Generating manager orders* for more details on this. You can apply a blueprint, then the planned buildings will be built as your dwarves fulfill the orders.

6.3.12 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 in the manager orders screen. You might also 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.

6.3.13 Tips and tricks

- 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 at the same time that you build adjacent walls. 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!

6.3.14 Caveats and limitations

- 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! If you run into any, please report them at the DFHack issue tracker so they can be addressed.

6.3.15 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 relatively 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 players 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.

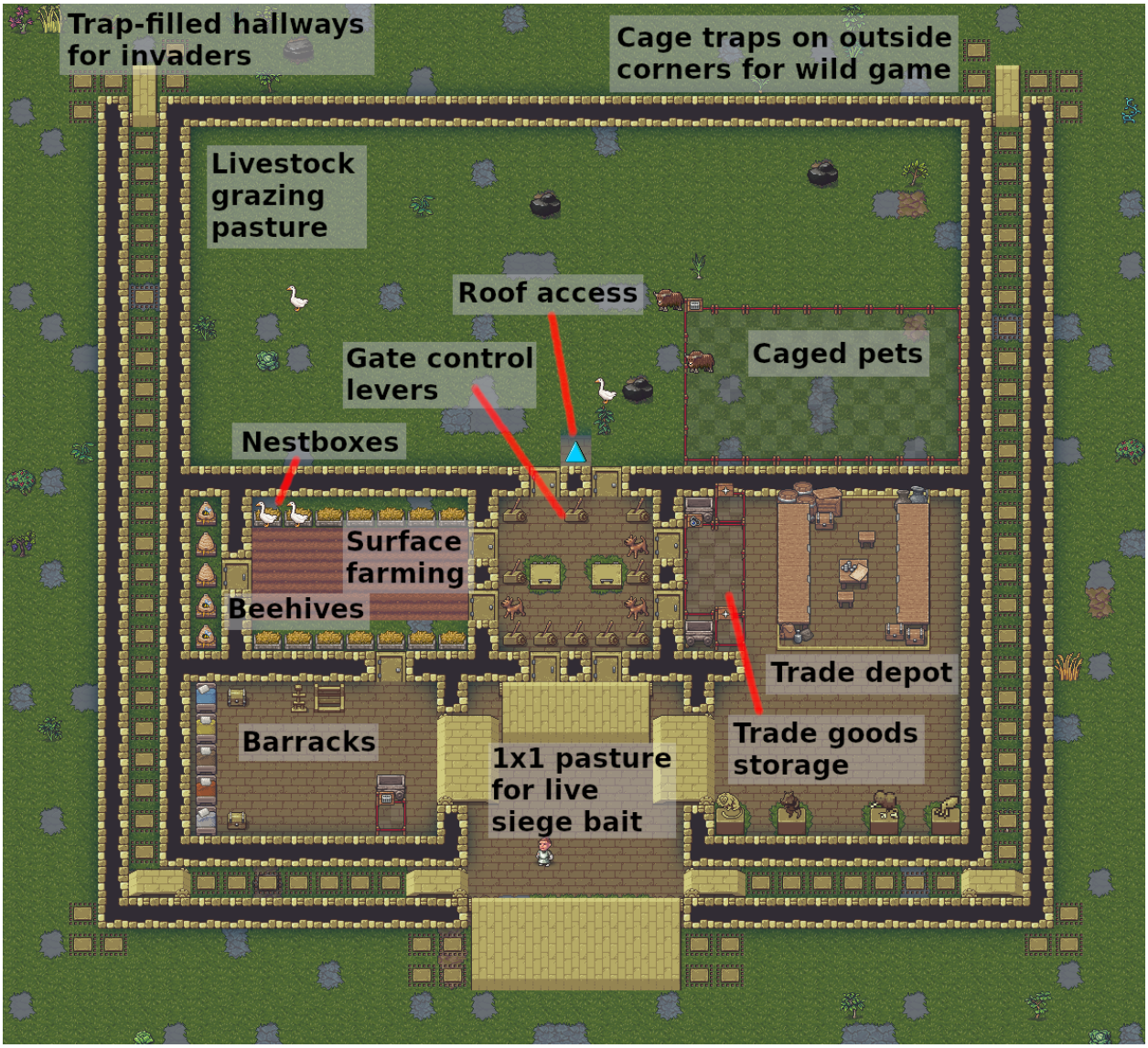
Tip

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.

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 linked to control levers, 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, and building

starting workshops with a `#build` blueprint can all be done with a single command. Bundling blueprints with `#meta` blueprints reduced the number of steps in Dreamfort from 100 to about 25, 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 *gui/quickfort* blueprint load dialog or `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 by toggling the "Hidden" setting in *gui/quickfort* or by passing the `--hidden` option to `quickfort list` 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 *gui/quickfort* and the `quickfort list` command. 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.

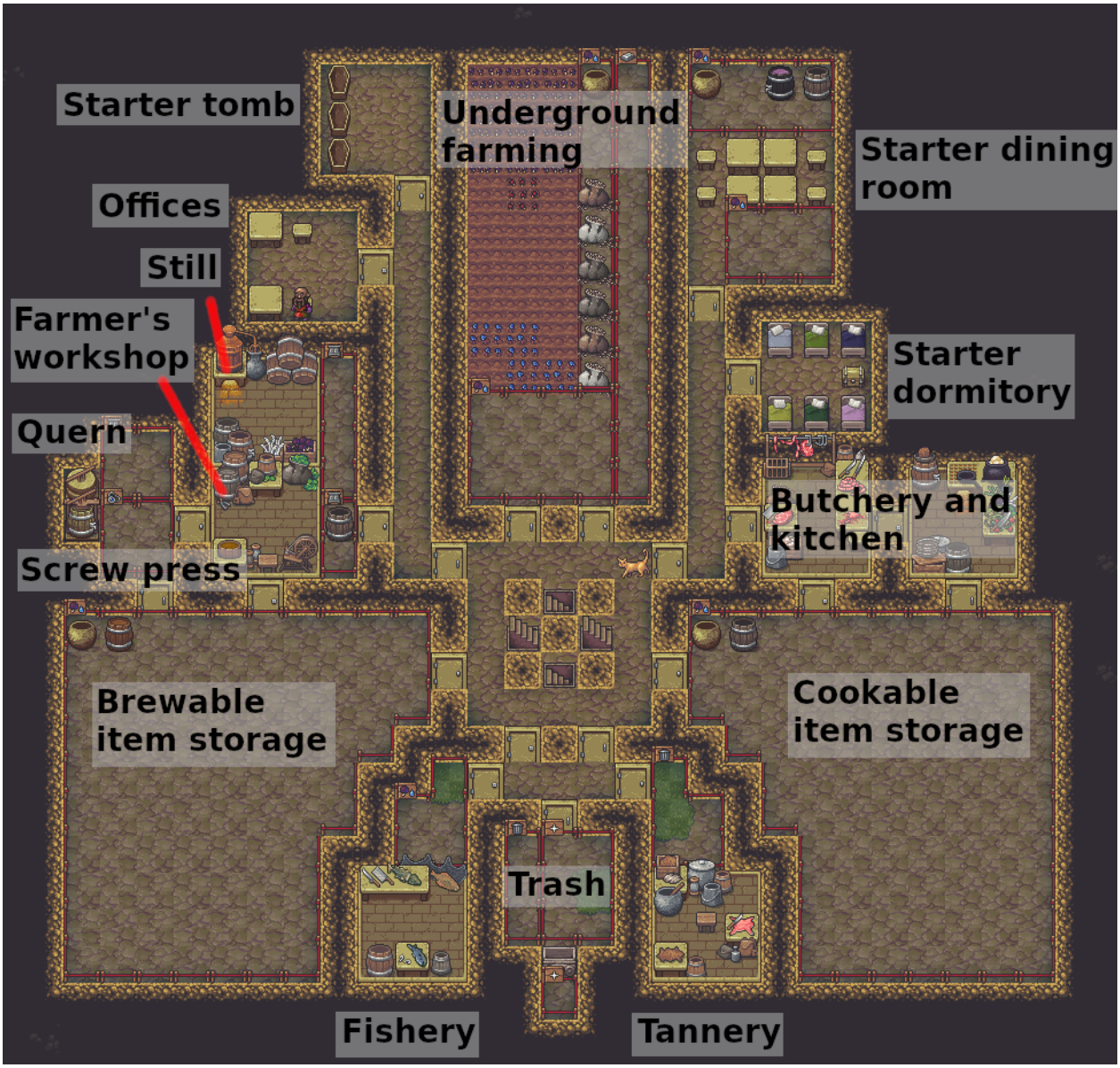
Messages are displayed after a blueprint is applied. Good things to include in messages are:

- The name of the next blueprint to apply and when to run it
- Whether orders should be generated 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 rottables can accumulate 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 with the `take_from` and `give_to` properties.

The farming level starts doing interesting things with stockpiles in its `#place` blueprints. Note the careful customiza-

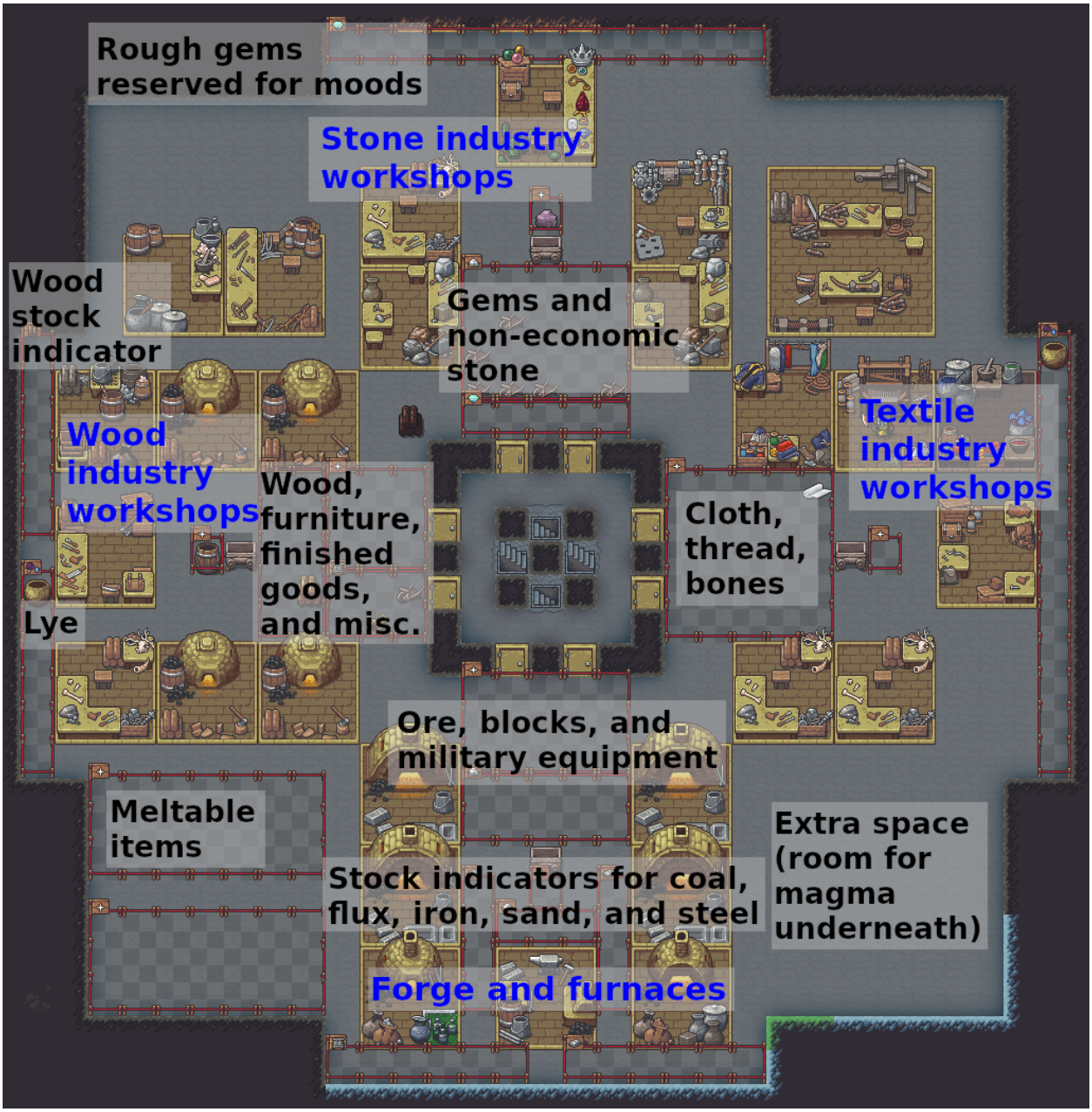
tion of the food stockpiles and the stockpile chains set up with the `take_from` and `give_to` properties. For example, the "Seeds" stockpile is set to `link_only=true` and the "Seeds feeder" stockpile has `containers=0` and `give_to=Seeds`. This minimizes container churn for the common task of seed recovery. When finding the named stockpiles to link, quickfort will search the other stockpiles created in the same blueprint first. If no stockpiles by that name are found, then existing stockpiles/workshops are searched. This is how many of the stockpiles on this level are configured to take from the starter stockpiles on the surface.

Tip

Quantum stockpiles are super easy to define, if you want to use them.

Hauling routes are notoriously fiddly to set up by hand, but they can be easily automated with blueprints. Check out the Southern area of the `#place` and `#build` blueprints for how the quantum refuse dump is configured with simple `take_from` and `route` properties attached to the track stop.

The industry level: advanced linking



The industry level is densely packed and has more intracate stockpile and hauling route configuration.

Tip

Name things.

In order to be a target for a stockpile or workshop link, the stockpile or building must have a name. That's not the only reason you should give things names, though. The game is just much easier to play when stockpiles and key buildings have descriptive names. Which lever controls the bridge named "Right outer gate"? You can click on that bridge, click on "show linked buildings", zoom to the lever, and click on the lever. Or you can scan your mouse over the levers and

click on the lever with the same name as the target bridge. You can always edit names in-game, but blueprints are a great way to automate this task.

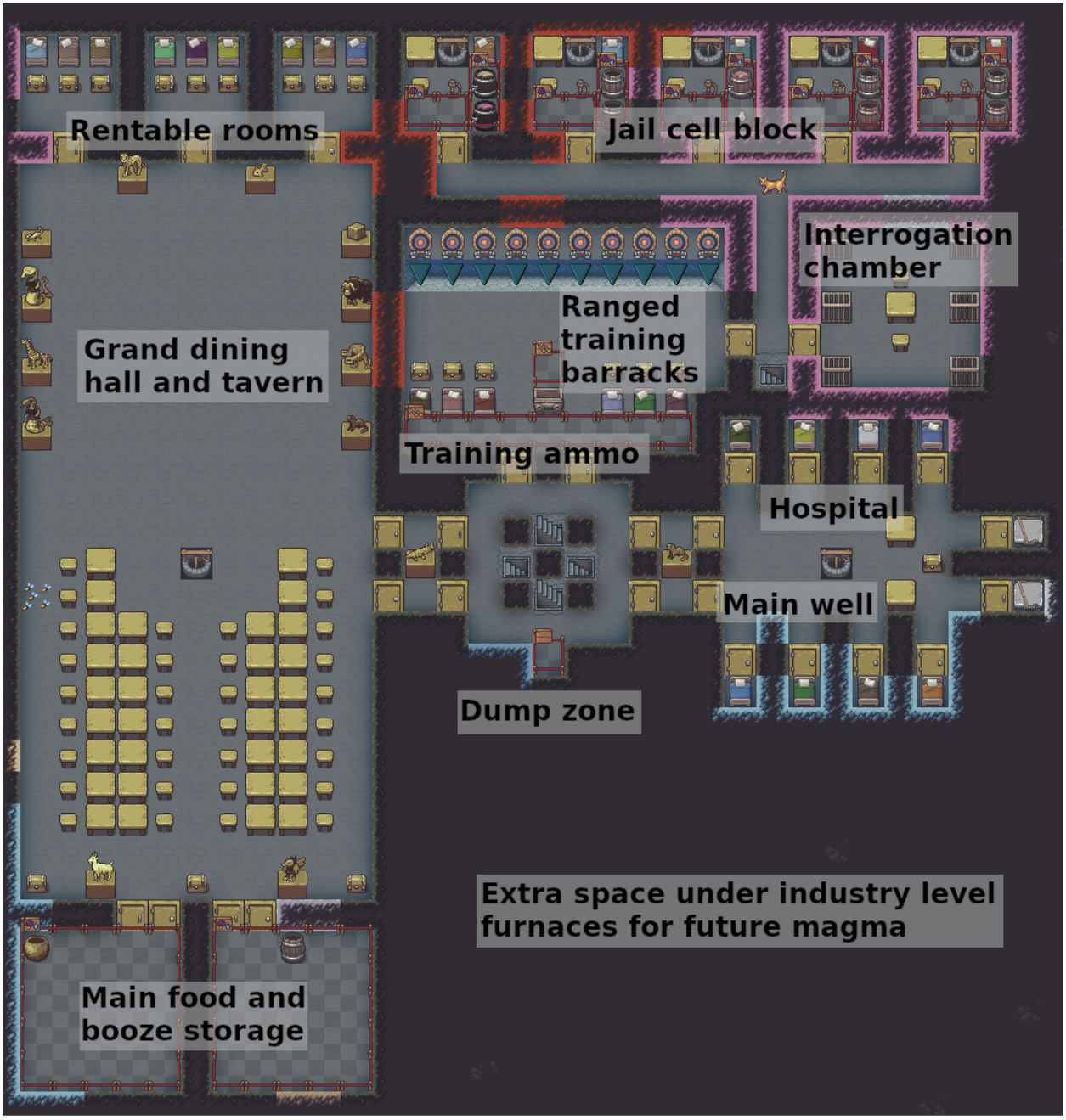
Tip

You can give to or take from multiple sources.

Some of the feeder stockpiles on this level are split up so that no one item type can fill the whole pile. The track stops that drive the quantum stockpiles have to take from all of them at once. When specifying multiple link targets that have spaces in their names, remember to surround the whole list with quotes. For example:

```
#build
trackstopW{name="Goods/wood dumper" take_from="Wood feeder,Goods feeder,Furniture feeder
↪" route="Goods/wood quantum"}
```

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 lower layers – scroll down on the blueprint spreadsheet). This works here because there is only a single column of higher-priority stairs for a dwarf to dig down to get below the lower-priority channels. If the dig area has multiple tiles exposed, it is harder to control dig order since a second dwarf may not have access to any higher-priority tiles and may start digging the lower-priority designations prematurely.

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 simultaneous digging of the two layers.

The guildhall level: avoiding smoothing issues

The goal of this level is to provide rooms for `locations` like guildhalls, libraries, and temples. The value of these rooms is very important, so we are likely to smooth and engrave everything. To smooth or engrave a wall tile, a dwarf has to be adjacent to it, and since some furniture, like statues, block dwarves from entering a tile, where you put them

affects what you can access.

Tip

Don't put statues in corners unless you want to smooth everything first.

In the guildhall level, the statues are placed so as not to block any wall corners. This gives the player freedom for choosing when to smooth. If a statue blocks a corner, or if a line of statues blocks a wall segment, it forces the player to smooth before building the statues. Otherwise they have to bother with temporarily removing statues to smooth the walls behind them.

The suites level: balance of flexibility

In designing this level, we needed to choose between two approaches:

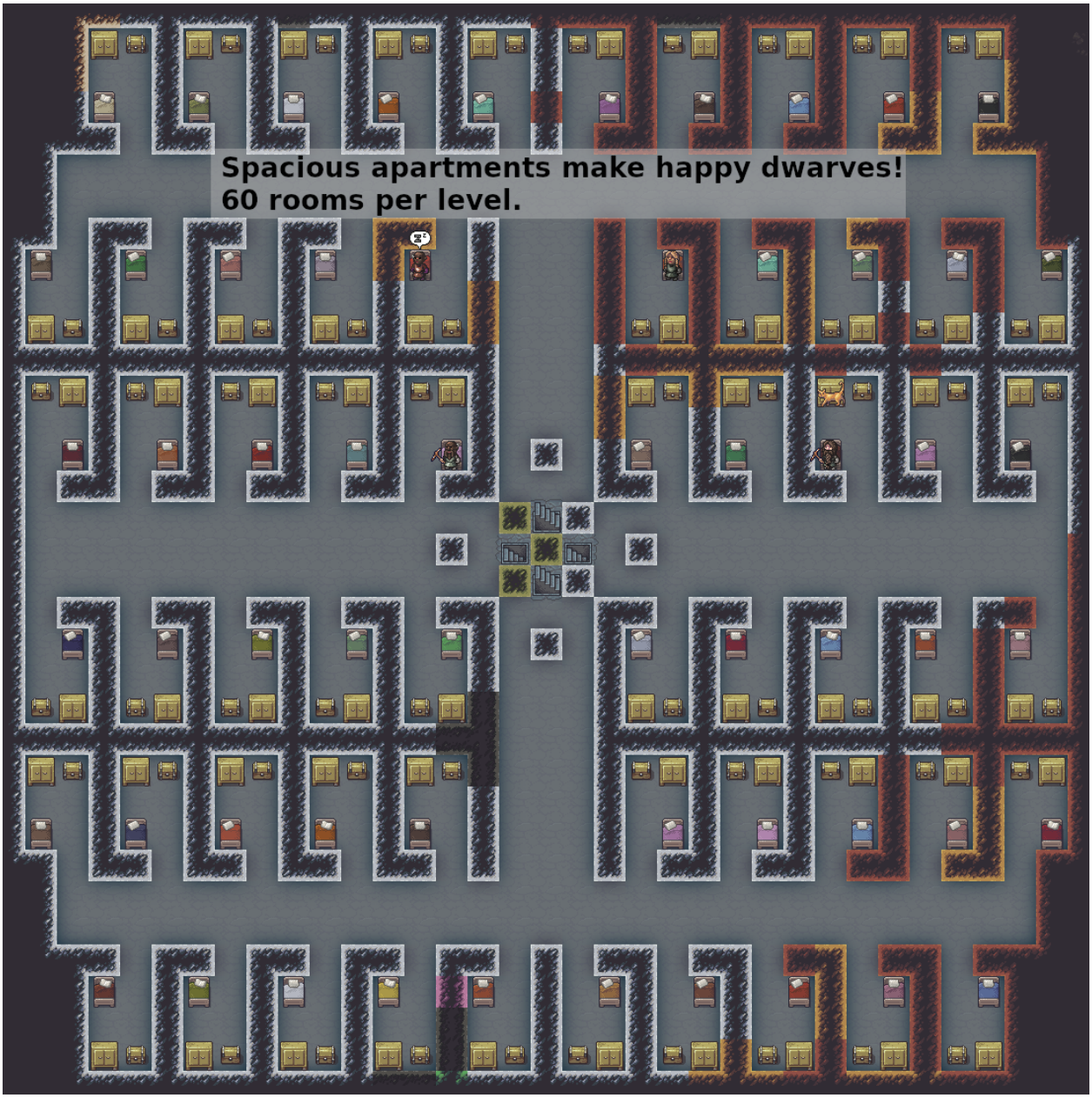
1. Create rooms with specific, pre-determined purposes, laying out furniture and zoning with appropriate types
2. Lay out each room the same so each can serve any purpose

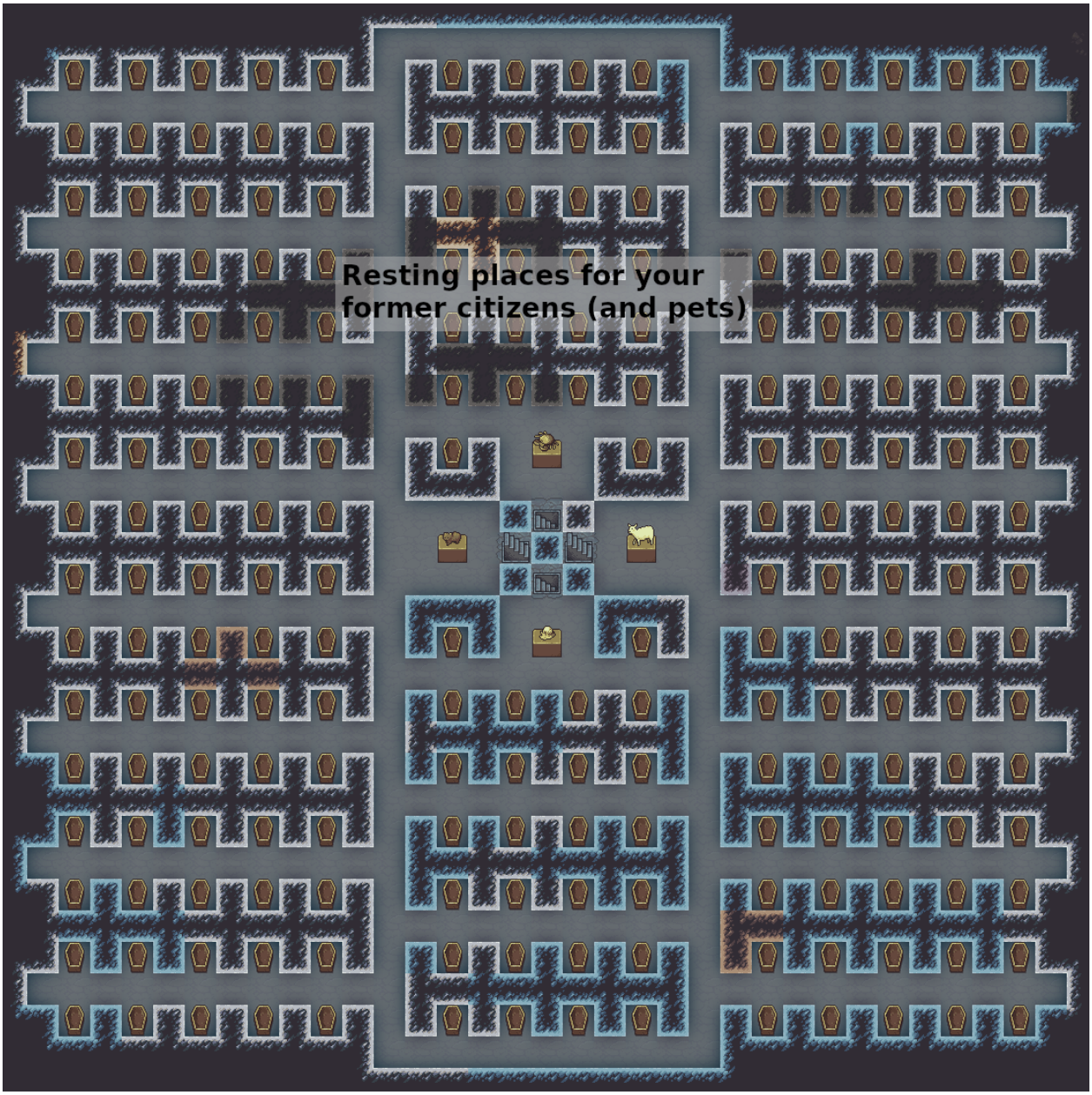
Each has pros and cons. The first option reduces more toil by pre-creating the zones. If we go this route, we can also auto-assign the rooms to the various roles (if they exist when the blueprint is run). Each room can be customized for its intended purpose: offices can look like offices, bedrooms could look like bedrooms, and so on. However, if the needs of the fort don't correspond to the pre-determined layouts, or if the needs of the fort *change* significantly, then the blueprint can become more of a hinderance than a help.

As you can see from the screenshot, we went with option 2. The ability to re-zone arbitrarily to meet changing noble needs was too great of a benefit to ignore. The downside, of course, is that you have to zone and assign your own rooms. However, as soon as you gain a barony or a duchy, you'd be doing that anyway with option 1.

With option 2, if you need a "better" bedroom, you'd just expand the zone to cover the neighboring "unit". Satisfying the monarch is also simple: plop down a new suites level and assign each block of 4 rooms to one zone. four units for the bedroom, four for the office, four for the dining hall, and four for the tomb. Smooth and engrave and you're done. Of course, more asthetic-minded players are always free to design custom areas too. These blueprints are designed to be functional more than beautiful.

The beds and crypt levels: copy and paste and repeat





The apartments and crypt blueprints are straightforward, other than the sheer number of zones. Copy-paste in Google Sheets was used heavily here. 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 three z-levels at once.

That's it! I hope this guide was useful to you. Please leave feedback on the forums or on the DFHack Discord server if you have ideas on how this guide (or the dreamfort blueprints themselves) can be improved!

6.3.16 Links

Quickfort links:

- *Quickfort command 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.

6.3.17 Appendix: Symbols and properties

#dig mode reference

Symbol	Meaning
`d`	dig (mine out walls but leave the floors)
`h`	channel (empty tile with a ramp in the z-level below)
`u`	up stair
`j`	down stair
`i`	up/down stair
`r`	ramp (produces empty tile in the z-level above)
`z`	remove up stairs/ramps
`t`	chop trees
`p`	gather plants
`s`	smooth walls or floors
`e`	engrave smoothed walls or floors
`F`	carve fortification
`T`	carve track
`v`	toggle whether engraving details are visible
`M`	toggle marker (called "blueprints" by the DF interface)
`n`	remove construction
`x`	remove designation
`bc`	claim items on this tile
`bf`	forbid items on this tile
`bm`	melt items on this tile
`bM`	remove melt flag from items on this tile
`bd`	dump items on this tile
`bD`	remove dump flag from items on this tile

continues on next page

Table 1 – continued from previous page

Symbol	Meaning
bh	hide items on this tile
bH	unhide items on this tile
oh	set high traffic
on	set normal traffic
ol	set low traffic
or	set restricted traffic
trackN	carve track that extends to the north
trackS	carve track that extends to the south
trackE	carve track that extends to the east
trackW	carve track that extends to the west
trackNS	carve track that extends to the north and south
trackEW	carve track that extends to the east and west
trackNE	carve track that extends to the north and east (corner)
trackNW	carve track that extends to the north and west (corner)
trackSE	carve track that extends to the south and east (corner)
trackSW	carve track that extends to the south and west (corner)
trackNSE	carve track that extends to the north, south, and east (tee)
trackNSW	carve track that extends to the north, south, and west (tee)
trackNEW	carve track that extends to the north, east, and west (tee)
trackSEW	carve track that extends to the south, east, and west (tee)
trackNSEW	carve track that extends in all directions (cross)

You can carve a track over an existing natural ramp to allow a minecart to safely traverse z-levels. You can write trackramp<dir> instead of track<dir> (e.g. trackrampSW) for clarity in blueprints where this is the intention. The actual designation produced by track<dir> and trackramp<dir> is identical.

Any of the symbols can be prefixed by any number of marker specifications:

Marker	Meaning
mb	"blueprint" mode
mw	warm dig
md	damp dig

For example, a 10x10 block of standard dig tiles with warm + damp markers would be mwmdd(10x10).

#zone mode reference

Symbol	Type	Properties
m	meeting area	
b	bedroom	
h	dining hall	
n	pen/pasture	
p	pit/pond	pond: if set to true, then the zone is a pond
w	water source	
j	dungeon	
f	fishing	
s	sand	
o	office	
D	dormitory	
B	barracks	
a	archery range	shoot_from: can be any of: west, left, east, right, north, top, south, or bottom. defaults to west.
d	garbage dump	
t	animal training	
T	tomb	pets: if set to true then pets are allowed. citizens: if set to false then citizens are not allowed.
g	gather/pick fruit	pick_trees, pick_shrubs, gather_fallen: all are set to true by default. set to false to disable.
c	clay	

In addition to the type-specific properties listed above, all zones support the following properties:

Property	Description
name	the name of the zone
activ	if set to false then the zone is deactivated
assig	if set to the name of a noble position (like manager or bookkeeper or sheriff) then the zone will be assigned to the unit appointed to the indicated noble position (if any). if the fort has progressed to the point that the sheriff role has been replaced by the captain of the guard, then assigned_unit=sheriff will be interpreted as assigned_unit=captain_of_the_guard.
locat	the type of location to create and attach the zone to: one of: hospital, guildhall, tavern, library, or temple. To attach multiple zones to the same location, specify a label after the location type and use the same label for all attached zones. For example: location=tavern/mainpub.
allow	(only if location is also set) sets the access restriction for the attached location: one of: visitors, residents, citizens, or members. defaults to visitors.
profe	(only if location=guildhall) sets the profession of the guildhall. See possilbe values with :lua @df.profession. For example: profession=metalsmith.
desir	(only if the location is set to the relevant type) sets the desired number of stocked items for the attached location. See the table below for details.

Here are the desired items that can be specified for each location type:

Location type	Properties
tavern	`desired_goblets` (defaults to `10`), `desired_instruments` (defaults to `5`)
hospital	`desired_splints` (defaults to `5`), `desired_thread` (defaults to `5`), `desired_cloth` (defaults to `5`), `desired_crutches` (defaults to `5`), `desired_powder` (defaults to `5`), `desired_buckets` (defaults to `2`), `desired_soap` (defaults to `5`)
library	`desired_paper` (defaults to `10`)
temple	`desired_instruments` (defaults to `5`)

#place mode reference

The symbol or symbols used to declare a stockpile determine which item categories are enabled by default as well as which stockpile containers (e.g. bins/barrels/wheelbarrows) are assigned to the stockpile by default.

Symbol	Type	Containers
`a`	animal	
`f`	food	barrels
`u`	furniture	
`n`	coins	bins
`y`	corpses	
`r`	refuse	
`s`	stone	wheelbarrows
`w`	wood	
`e`	gem	bins
`b`	bars and blocks	bins
`h`	cloth	bins
`l`	leather	bins
`z`	ammo	bins
`S`	sheets	bins
`g`	finished goods	bins
`p`	weapons	bins
`d`	armor	bins
`c`	custom	

All stockpiles support the following properties:

Property	Description
`name`	the name of the stockpile
`take_from`	comma-separated list of names or building ids of stockpiles or workshops that the stockpile takes from
`give_to`	comma-separated list of names or building ids of stockpiles or workshops that the stockpile gives to
`links_only`	if set to `true` then the stockpile will only take from links
`barrels`	the number of desired barrels
`bins`	the number of desired bins
`wheelbarrow`	the number of desired wheelbarrows
`containers`	sets the desired numbers for barrels, bins, and wheelbarrows. this is most useful to set all desired container counts to 0.
`quantum`	if set to `true`, then sets `containers=0 links_only=true`
`automelt`	if set to `true`, enables the corresponding *logistics* feature
`autotrade`	if set to `true`, enables the corresponding *logistics* feature
`autodump`	if set to `true`, enables the corresponding *logistics* feature
`autotrain`	if set to `true`, enables the corresponding *logistics* feature

Note that specifying building IDs in `take_from` or `give_to` lists is primarily useful when dynamically generating *quickfort* blueprints and applying them via the API. You will not generally know the ID of a stockpile or building when writing a blueprint by hand or when preparing a blueprint to apply in a different fort.

#build mode reference

In addition to the type-specific properties listed in the symbol table below, all building types accept the `name` property.

Moreover, all workshops and furnaces (both called "workshops" below) accept the following "profile" properties:

Property	Description
`max_gener`	the maximum number of general workorders that the workshop will accept
`take_from`	comma-separated list of names or building ids of stockpiles that the workshop takes from
`give_to`	comma-separated list of names or building ids of stockpiles that the workshop gives to
`labor`	comma-separated list of labors that should be enabled for the workshop. all unlisted labors for the workshop will be disabled.
`labor_mas`	comma-separated list of labors that should be disabled for the workshop. all unlisted labors for the workshop will be left enabled. if both the `labor` and `labor_mask` properties are specified, the `labor` property takes precedence.
`min_skill`	the minimum skill rating for units that perform jobs at the workshop
`max_skill`	the maximum skill rating for units that perform jobs at the workshop

For the `labor` and `labor_mask` properties, you can use either the labor ID or the readable caption string. You can see both options by running:

```
:lua for idx,name in ipairs(df.unit_labor) do cap=df.unit_labor.attrs[idx].caption if␣
↪cap then print(('%22s %25s'):format(name, cap)) end end
```

For example, you can specify either `BONE_CARVE` or `"Bone Carving"` (it contains a space – don't forget the surrounding quotes!) to indicate the bone carving labor.

Likewise, for the `min_skill` and `max_skill` properties, you can specify either the skill rating ID or the readable caption string. You can see both options by running:

```
:lua for idx,name in ipairs(df.skill_rating) do cap=df.skill_rating.attrs[idx].caption␣
↪if cap then print(('%22s %25s'):format(name, cap)) end end
```

Symbol	Type	Properties
a	armor stand	
b	bed	
c	seat	
n	burial receptacle	
d	door	
x	floodgate	
H	floor hatch	
W	wall grate	
G	floor grate	
B	vertical bars	
~b	floor bars	
f	cabinet	
h	container	
r	weapon rack	
s	statue	
~s	slab	
t	table	
g	bridge (retracting)	
gw	bridge (raises to north)	
gd	bridge (raises to east)	
gx	bridge (raises to south)	
ga	bridge (raises to west)	
l	well	
y	glass window	
Y	gem window	
D	trade depot	
Msu	screw pump (pumps from north)	
Msk	screw pump (pumps from east)	
Msm	screw pump (pumps from south)	
Msh	screw pump (pumps from west)	
Mw	water wheel (vertical)	
Mws	water wheel (horizontal)	
Mg	gear assembly	
Mh	horizontal axle (east-west)	
Mhs	horizontal axle (north-south)	
Mv	vertical axle	
rollerNS	roller (pushes to south)	speed: one of 50000, 40000, 30000, 20000, or 10000. defaults to 50000.
rollerEW	roller (pushes to west)	speed
rollerSN	roller (pushes to north)	speed
rollerWE	roller (pushes to east)	speed
S	support	
m	animal trap	
v	restraint	
j	cage	
A	archery target	
R	traction bench	
N	nest box	

Symbol	Type	Properties
~h	hive	if do_install is set to true then a bee colony will be installed. if do_gathe
~a	offering place	
~c	bookcase	
F	display furniture	
p	farm plot	if seasonal_fertilize is set to true then the plots will be configured for s
o	paved road	
O	dirt road	
k	vermin catcher	
we	leather works	
wq	quern	
wM	millstone	
wo	loom	
wk	clothier	
wb	bowyer	
wc	carpenter	
wf	metalsmith	
wv	magma forge	
wj	jeweler	
wm	stoneworker	
wu	butcher	
wn	tanner	
wr	craftsdwarf	
ws	siege workshop	
wt	mechanic	
wl	still	
ww	farmer	
wz	kitchen	
wh	fishery	
wy	ashery	
wd	dyer	
wS	soap maker	
wp	screw press	
ew	wood furnace	
es	smelter	
el	magma smelter	
eg	glass furnace	
ea	magma glass furnace	
ek	kiln	
en	magma kiln	
ib	ballista	
ic	catapult	
Cw	wall	
Cf	floor	
Cr	ramp	
Cu	up stair	
Cd	down stair	
Cx	up/down stair	
CF	fortification	
trackstop	track stop (no dumping)	friction: one of 50000, 10000, 500, 50, or 10. defaults to 50000. take_fr
trackstopN	track stop (dump to north)	friction, take_from, route

Symbol	Type	Properties
trackstopS	track stop (dump to south)	friction, take_from, route
trackstopE	track stop (dump to east)	friction, take_from, route
trackstopW	track stop (dump to west)	friction, take_from, route
Ts	stone-fall trap	
Tw	weapon trap	
Tl	lever	
Tp	pressure plate	
Tc	cage trap	
TS	upright spear/spike	
trackN	track to the N	
trackS	track to the S	
trackE	track to the E	
trackW	track to the W	
trackNS	track to the N, S	
trackEW	track to the E, W	
trackNE	track corner to the N, E	
trackNW	track corner to the N, W	
trackSE	track corner to the S, E	
trackSW	track corner to the S, W	
trackNSE	track tee to the N, S, E	
trackNSW	track tee to the N, S, W	
trackNEW	track tee to the N, E, W	
trackSEW	track tee to the S, E, W	
trackNSEW	track cross	
trackrampN	track ramp to the N	
trackrampS	track ramp to the S	
trackrampE	track ramp to the E	
trackrampW	track ramp to the W	
trackrampNS	track ramp to the N, S	
trackrampEW	track ramp to the E, W	
trackrampNE	track ramp corner to the N, E	
trackrampNW	track ramp corner to the N, W	
trackrampSE	track ramp corner to the S, E	
trackrampSW	track ramp corner to the S, W	
trackrampNSE	track ramp tee to the N, S, E	
trackrampNSW	track ramp tee to the N, S, W	
trackrampNEW	track ramp tee to the N, E, W	
trackrampSEW	track ramp tee to the S, E, W	
trackrampNSEW	track ramp cross	

#burrow mode reference

Symbol	Meaning	Properties
a	add	`name`: if set, will add to an existing burrow of this name. `create`: if set to `true`, will create a burrow with the specified `name` if it doesn't already exist. `civalert`: if set to `true`, will register this burrow with *gui/civ-alert* if no burrow has already been registered. `autochop_clear`: if set to `true`, register the burrow with *autochop* so that all trees in the burrow are immediately chopped down, regardless of how many logs are in stock. `autochop_chop`: if set to `true`, register the burrow with `autochop` so that woodcutting activity is constrained to this burrow (and others marked for `chop`).
e	erase	`name`: if set, will only affect the first burrow of the given name. if not set, will affect all burrows that cover the given tiles.

CHAPTER

SEVEN

DFHACK DEVELOPMENT GUIDE

These are pages relevant to people developing for DFHack.

7.1 DFHack development overview

This page provides an overview of DFHack components. If you are looking to develop a tool for DFHack, developing a script or plugin is likely the most straightforward choice.

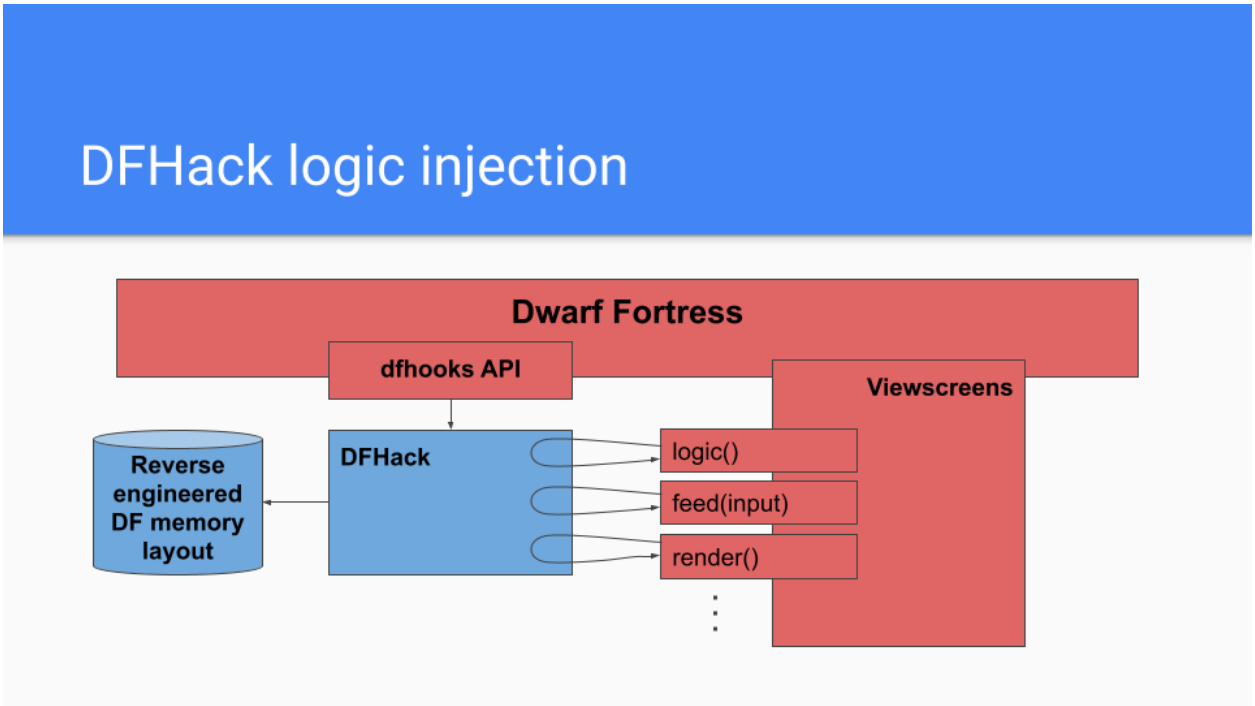
Other pages that may be relevant include:

- *Building DFHack*
- *How to contribute to DFHack*
- *DFHack documentation system*
- *Licenses*

Contents

7.1.1 Architecture diagrams

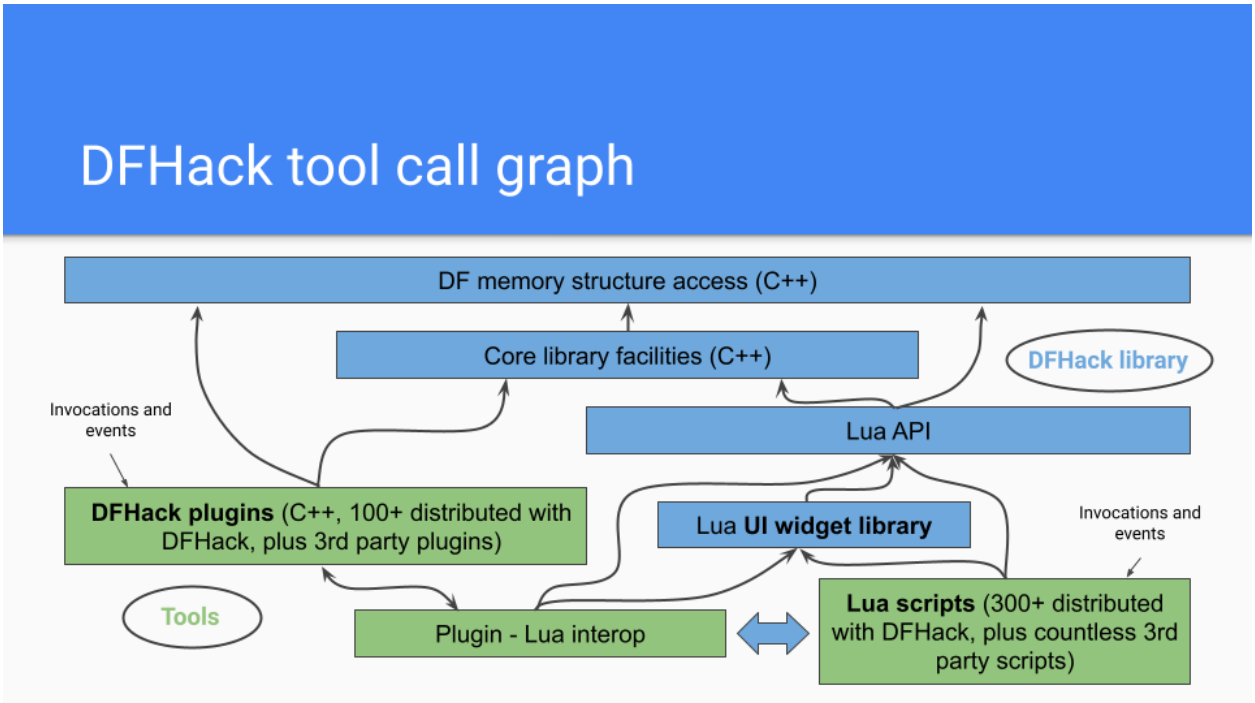
These two diagrams give a very high level overview of where DFHack fits into the DF call structure and how the pieces of DFHack itself fit together:



When DF loads, it looks for a "dfhooks" library file (named appropriately per platform, e.g. `libdfhooks.so` on Linux). DFHack provides this library file, and DF calls the API functions at specific points in its initialization code and main event loop.

In addition, DFHack can "interpose" the virtual methods of DF classes. In particular, it intercepts calls to the interface functions of each DF viewscreen class to provide *overlay* functionality.

The dfhooks API is defined in DF's open source component `g_src`: https://github.com/Putnam3145/Dwarf-Fortress–libgraphics–/blob/master/g_src/dfhooks.h



DF memory layout is encoded in the xml files of the df-structures repository. These XML files are converted into C++ header files during the build process.

The functionality of the DFHack core library is grouped by *Modules* that access DF memory according to the defined structures.

The Lua API layer makes DFHack core facilities available to Lua scripts. Both the C++ and Lua APIs have a library of convenience functions, though only the Lua API is *well-documented*. Notably, the entire *UI widget library* is Lua-only, though C++ plugins can easily access it via the plugin-Lua interop layer.

7.1.2 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 is also 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.

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.

7.1.3 Scripts

DFHack scripts are written in Lua, with a *well-documented library*. Referring to existing scripts as well as the API documentation is very 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.

7.1.4 Core

The *DFHack core* has a variety of low-level functions. It is responsible for implementing the dfhooks API that DF calls, it provides a console, and it provides an interface for plugins and scripts to interact with DF.

7.1.5 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.

Most modules are also *exposed to Lua*, although some functions (and some entire modules) are currently only available in C++.

7.1.6 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.

7.2 Building DFHack

Those seeking to compile the source code for DFHack, including core and plugins, can refer to the following help pages.

7.2.1 Dependencies

DFHack is meant to be installed into an **existing DF folder**, so ensure that one is ready.

Contents

Overview of Dependencies

This section provides an overview of system dependencies that DFHack relies on. See the platform-specific sections later in this document for specifics on how to install these dependencies.

DFHack also has several dependencies on libraries that are included in the repository as Git submodules, which require no further action to install.

System dependencies

- CMake (v3.21 or newer is recommended)
- build system (e.g. gcc & ninja, or Visual Studio)
- Perl 5 (for code generation) * XML::LibXML * XML::LibXSLT
- Python 3 (for *documentation*) * Sphinx
- Git (required for contributions)
- ccache (**optional**, but strongly recommended to improve build times)
- OpenGL headers (**optional**: to build *stonesense*)

- zlib (compression library used for *xlsxreader* -> *quickfort*)

Perl packages

- XML::LibXML
- XML::LibXSLT

The Perl packages are used in code generation. DF memory structures are represented as XML in DFHack's source tree. During the configuration process (cmake) the xml files are converted into C++ headers and Lua wrappers for use by plugins and scripts.

Python packages

- Sphinx (required to build the *documentation*)

Installing Dependencies

Linux

Here are some package install commands for various distributions:

- On Arch Linux:

```
pacman -Sy gcc cmake ccmake ninja git dwarffortress zlib perl-xml-libxml perl-xml-
↪libxslt
```

  - The `dwarffortress` package provides the necessary SDL packages.
  - 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 libsdl2-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 SDL2-devel perl-core perl-XML-
↪LibXML perl-XML-LibXSLT ruby
```

To build DFHack, you need GCC 10 or newer. 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).

Distributing binaries compiled with newer GCC versions may result in compatibility issues for players with older GCC versions. This is why DFHack builds distributables with GCC 10, which is the same GCC version that DF itself is compiled with.

Before you can build anything, you'll also need `cmake`. It is advisable to also get `ccmake` (the interactive configuration interface) on distributions that split the cmake package into multiple parts. As mentioned above, `ninja` is recommended as the build system (many distributions call this package `ninja-build`), but `make` also works.

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.

Windows

DFHack must be built with the Microsoft Visual C++ 2022 toolchain (aka MSVC v143) for ABI compatibility with Dwarf Fortress v50.

With Chocolatey

Many of the dependencies are simple enough to download and install via the chocolatey package manager on the command line.

Here are some package install commands:

```
choco install cmake
choco install ccache
choco install strawberryperl
choco install python
choco install sphinx

# Visual Studio
choco install visualstudio2022community --params "--add Microsoft.VisualStudio.Workload.
↪NativeDesktop --includeRecommended"
# OR
# Build Tools for Visual Studio
choco install visualstudio2022buildtools --params "--add Microsoft.VisualStudio.Workload.
↪NativeDesktop --includeRecommended"
```

If you already have Visual Studio 2022 or the Build Tools installed, you may need to modify the installed version to include the workload components listed in the manual installation section, as chocolatey will not amend the existing install.

Manually

If you prefer to install manually rather than using Chocolatey, details and requirements are as below. If you do install manually, **ensure that your PATH variable is updated** to include the install locations for all tools. This can be edited from `Control Panel -> System -> Advanced System Settings -> Environment Variables`.

CMake

You can get the Windows installer 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 the `XML::LibXML` and `XML::LibXSLT` packages installed. 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. 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` (path 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.

Python

See the Python website. Any supported version of Python 3 will work.

Sphinx

See the Sphinx website.

Visual Studio

The required toolchain can be installed as a part of either the Visual Studio 2022 IDE or the Build Tools for Visual Studio 2022. If you already have a preferred code editor, the Build Tools will be a smaller install. You may need to log into (or create) a Microsoft account in order to download Visual Studio.

Build Tools [Without Visual Studio]

Click Build Tools for Visual Studio 2022 and you will be prompted to login to your Microsoft account. Then you should be redirected to a page with various download options with 2022 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 want to select the most up-to-date version – as of writing this is "Build Tools for Visual Studio 2022 (version 17.4)". "LTSC" is an extended support variant and is not required for our purposes.

When installing, select the "Desktop Development with C++" workload and ensure that the following are checked:

- MSVC v143 - VS 2022 C++ x64/x86 build tools
- C++ CMake tools for Windows
- At least one Windows SDK (for example, Windows 11 SDK 10.0.22621).

macOS

NOTE: this section is currently outdated. Once DF itself can build on macOS again, we will match DF's build environment and update the instructions here.

DFHack is easiest to build on macOS with exactly 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.

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.
6. Install Python dependencies
   - You can choose to use a system Python 3 installation or any supported version of Python 3 from python.org.
   - Install Sphinx

7.2.2 Compilation

DFHack builds are available for all supported platforms; see *Installing* for installation instructions. If you are a DFHack end-user, modder, or plan on writing scripts [lua] (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

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 *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.

All Platforms

Before you can compile the code you'll need to configure your build with cmake. Some IDEs can do this for you, but it's more common to do it from the command line. Windows developers can refer to the Windows section below for batch files that can be used to avoid opening a terminal/command-prompt.

You should seek cmake's documentation online or via `cmake --help` to see how the command works. See the *Build Options* page for help finding the DFHack build options relevant to you.

Before compiling code, you'll of course need code to compile. This **will include** the submodules, so be sure you've read the section about getting the code.

Linux

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.

Windows

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.

Visual Studio IDE

After running the CMake generate script you will have a new folder called VC2022 or VC2022_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 2022.

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`.

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. Modern Windows terminal emulators such as Cmder or Windows Terminal provide a better experience by providing more scrollback and larger window sizes.

macOS

NOTE: this section is currently outdated. Once DF itself can build on macOS again, we will match DF's build environment and update the instructions here.

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.

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.

Notes for GCC 8+ or OS X 10.10+ users

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
```

Windows cross compiling from Linux (running DF inside docker)

You can use docker to build DFHack for Windows. These instructions were developed on a Linux host system.

Step 1: prepare a build container

On your Linux host, install and run the docker daemon and then run these commands:

```
xhost +local:root
docker run -it --env="DISPLAY" --env="QT_X11_NO_MITSHM=1" --volume=/tmp/.X11-unix:/tmp/.
↪X11-unix --user buildmaster --name dfhack-win ghcr.io/dfhack/build-env:msvc
```

The `xhost` command and `--env` parameters are there so you can eventually run Dwarf Fortress from the container and have it display on your host.

Step 2: build DFHack

The `docker run` command above will give you a shell prompt (as the `buildmaster` user) in the container. Inside the container, run the following commands:

```
git clone https://github.com/DFHack/dfhack.git
cd dfhack
git submodule update --init
cd build
dfhack-configure windows 64 Release
dfhack-make
```

Inside the `dfhack-*` scripts there are several commands that set up the wine server. Each invocation of a Windows tool will cause wine to run in the container. Preloading the wineserver and telling it not to exit will speed configuration and compilation up considerably (approx. 10x). You can configure and build DFHack with regular `cmake` and `ninja` commands, but your build will go much slower.

Step 3: copy Dwarf Fortress to the container

First, create a directory in the container to house the Dwarf Fortress binary and assets:

```
mkdir ~/df
```

If you can just download Dwarf Fortress directly into the container, then that's fine. Otherwise, you can do something like this in your host Linux environment to copy an installed version to the container:

```
cd ~/.steam/steam/steamapps/common/Dwarf\ Fortress/
docker cp . dfhack-win:df/
```

Step 4: install DFHack and run DF

Back in the container, run the following commands:

```
cd dfhack/build
cmake .. -DCMAKE_INSTALL_PREFIX=/home/buildmaster/df
ninja install
cd ~/df
wine64 "Dwarf Fortress.exe"
```

Other notes

Closing your shell will kick you out of the container. Run this command on your Linux host when you want to reattach:

```
docker start -ai dfhack-win
```

If you edit code and need to rebuild, run `dfhack-make` and then `ninja install`. That will handle all the wineserver management for you.

Cross-compiling windows files for running DF in Steam for Linux

If you wish, you can use Docker to build just the Windows files to copy to your existing Steam installation on Linux.

Step 1: Get dfhack, and run the build script

Check out `dfhack` into another directory, and run the build script:

```
git clone https://github.com/DFHack/dfhack.git
cd dfhack
git submodule update --init --recursive
cd build
./build-win64-from-linux.sh
```

The script will mount your host's `dfhack` directory to docker, use it to build the artifacts in `build/win64-cross`, and put all the files needed to install in `build/win64-cross/output`.

If you need to run `docker` using `sudo`, run the script using `sudo` rather than directly:

```
sudo ./build-win64-from-linux.sh
```

Step 2: install dfhack to your Steam DF install

As the script will tell you, you can then copy the files into your DF folder:

```
# Optional -- remove the old hack directory in case we leave files behind
rm ~/.local/share/Steam/steamapps/common/"Dwarf Fortress"/hack
cp -r win64-cross/output/* ~/.local/share/Steam/steamapps/common/"Dwarf Fortress"/
```

Afterward, just run DF as normal.

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.

7.2.3 Build Options

Typical Options

- *Generator*
- *Install Location*
- *Build type*
- *Target architecture (32/64-bit)*
- *Library*
- *Testing*
- *Plugins*
- *Documentation*

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.

Typical usage may look like:

```
# Plugin development with updated documentation
cmake ./ -G Ninja -B builds/debug-info/ -DCMAKE_INSTALL_PREFIX=<path to DF> -DCMAKE_
↪BUILD_TYPE:string=RelWithDebInfo -DBUILD_DOCS:bool=ON -DBUILD_PLUGINS=1
```

(continues on next page)

(continued from previous page)

```
# Core DFHack only
cmake ../ -G Ninja -DCMAKE_INSTALL_PREFIX=<path to DF> -DCMAKE_BUILD_
↪TYPE:string=RelWithDebInfo -DBUILD_TESTS -DBUILD_DOCS:0 -DBUILD_PLUGINS=0
```

Modifying Build Options

You can typically run new cmake commands from your build directory to turn on/off options. Of course the generator selection is not something you can change, but the rest are.

Additionally, you can edit the build settings in CMakeCache.txt. You also have cmake's configuration utility `ccmake`.

Generator

For the uninitiated, the generator is what allows cmake to, of course, generate visual studio solution & project files, a makefile, or anything else. Your selection of generator comes down to preference and availability.

Visual Studio

To generate visual studio project files, you'll need to select a particular version of visual studio, and match that to your system's generator list viewed with `cmake --help`

example:

```
cmake .. -G "Visual Studio 17 2022"
```

Ninja

The generally preferred build system where available.

example:

```
cmake .. -G Ninja
```

Install Location

This is the location where DFHack will be installed.

Variable: `CMAKE_INSTALL_PREFIX`

Usage:

```
cmake .. -DCMAKE_INSTALL_PREFIX=<path to df>
```

The path to df will of course depend on your system. If the directory exists it is recommended to use `~/.dwarffortress` to avoid permission troubles.

Build type

This is the type of build you want. This controls what information about symbols and line numbers the debugger will have available to it.

Variable: `CMAKE_BUILD_TYPE`

Usage:

```
cmake .. -DCMAKE_BUILD_TYPE:string=RelWithDebInfo
```

Options:

- Release
- RelWithDebInfo

Target architecture (32/64-bit)

You can set this if you need 32-bit binaries or are looking to be explicit about building 64-bit.

Variable: `DFHACK_BUILD_ARCH`

Usage:

```
cmake .. -DDFHACK_BUILD_ARCH=32
```

Options:

- '32'
- '64' (default option)

Library

This will only be useful if you're looking to avoid building the library core, as it builds by default.

Variable: `BUILD_LIBRARY`

Usage:

```
cmake .. -DBUILD_LIBRARY:bool=OFF
cmake .. -DBUILD_LIBRARY=0
```

Testing

Regression testing will be arriving in the future, but for now there are only tests written in lua.

Variables:

- `BUILD_TESTING` (will build unit tests, in the future)
- `BUILD_TESTS` (installs lua tests)

Usage:

```
cmake .. -DBUILD_TESTS:bool=ON
cmake .. -DBUILD_TESTS=1
```

Plugins

If you're doing plugin development.

Variable: `BUILD_PLUGINS`

Usage:

```
cmake .. -DBUILD_PLUGINS:bool=ON
cmake .. -DBUILD_PLUGINS=1
```

Documentation

If you need to build documentation. Documentation can be built as HTML, and PDF, but there are also plain text files generated for in-game.

Variable: `BUILD_DOCS`

Usage:

```
cmake .. -DBUILD_DOCS:bool=ON
cmake .. -DBUILD_DOCS=1
```

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.

7.3 How to contribute to DFHack

Contents

7.3.1 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. 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:

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 or on Disord can easily get missed or forgotten.
- Work on reported problems will take priority over ideas or suggestions.

Code format

- Four space indents for C++. Never use tabs for indentation in any language.
- LF (Unix style) line terminators
- No 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.

7.3.2 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 (C++ api is rife)
- 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!

7.4 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

7.4.1 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.

7.4.2 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
========

.. 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
-----

``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>``
    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>`).

7.4.3 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):

```
-- A short description of my cool script.
```

The main help text for an external script needs to appear between two markers – `[====[` and `]====]`. The documentation standards above still apply to external tools, but there is no need to include backticks for links or monospaced fonts. Here is an 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

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.

7.4.4 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.

7.4.5 Building the documentation

Once the required dependencies are installed, there are multiple ways to run Sphinx to build the docs:

Using CMake

See our page on *build options*

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
```

7.4.6 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.

Template for new versions:

New Tools

New Features

Fixes

Misc Improvements

Documentation

API

Lua

Removed

7.4.7 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.

7.5 DFHack API concepts

7.5.1 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".

7.6 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

7.6.1 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. Fails with an error if the field is not found.

- `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.

Additionally, bitfields have a `whole` property, which returns the value of the bitfield as an integer.

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_identity` object.

All compound types (structs, classes, unions, and the global object) support:

- `type._fields`

  Contains a table mapping field names to descriptions of the type's fields, including data members and functions. Iterating with `pairs()` returns data fields in the order they are defined in the type. Functions and globals may appear in an arbitrary order.

  Each entry contains the following fields:

  - `name`: the name of the field (matches the `_fields` table key)
  - `offset`: for data members, the position of the field relative to the start of the type, in bytes
  - `count`: for arrays, the number of elements
  - `mode`: implementation detail. See `struct_field_info::Mode` in `DataDefs.h`.

  Each entry may also contain the following fields, depending on its type:

  - `type_name`: present for most fields; a string representation of the field's type
  - `type`: the type object matching the field's type; present if such an object exists (e.g. present for DF types, absent for primitive types)
  - `type_identity`: present for most fields; a lightuserdata pointing to the field's underlying `DFHack::type_identity` object
  - `index_enum`, `ref_target`: the type object corresponding to the field's similarly-named XML attribute, if present
  - `union_tag_field`, `union_tag_attr`, `original_name`: the string value of the field's similarly-named XML attribute, if present

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.

Enum types also support the `type.next_item(index)` function, which returns the next valid numeric value of the enum. It Returns the first enum value if `index` is greater than or equal to the max enum value.

Struct and class types with an instance-vector attribute in the XML also support:

- `type.find(key)`

  Returns an object from the instance vector that matches the key, where the field is determined by the 'key-field' specified in the XML.

- `type.get_vector()`

  Returns the instance vector e.g `df.item.get_vector() == df.global.world.items.all`

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.

7.6.2 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 tool state in the world savegame directory. It is intended for data that is world-dependent. Global state that is independent of the loaded world should be saved into a separate file named after the tool in the `dfhack-config/` directory.

Entries are associated with the current loaded site (fortress) and are identified by a string `key`. The data will still be associated with a fort if the fort is retired and then later unretired. Entries are stored as serialized strings, but there are convenience functions for working with arbitrary Lua tables.

- `dfhack.persistent.getSiteData(key[, default])`

  Retrieves the Lua table associated with the current site and the given string `key`. If `default` is supplied, then it is returned if the key isn't found in the current site's persistent data.

Example usage:

```
local state = dfhack.persistent.getSiteData('my-script-name', {somedata={}})
```

- `dfhack.peristent.getSiteDataString(key)`

  Retrieves the underlying serialized string associated with the current site and the given string `key`. Returns *nil* if the key isn't found in the current site's persistent data. Most scripts will want to use `getSiteData` instead.

- `dfhack.peristent.saveSiteData(key, data)`

  Persists the given `data` (usually a table; can be of arbitrary complexity and depth) in the world save, associated with the current site and the given `key`.

- `dfhack.persistent.saveSiteDataString(key, data_str)`

  Persists the given string in the world save, associated with the current site and the given `key`.

- `dfhack.persistent.deleteSiteData(key)`

  Removes the existing entry associated with the current site and the given `key`. Returns *true* if succeeded.

- `dfhack.persistent.getWorldData(key[, default])`
- `dfhack.peristent.getWorldDataString(key)`
- `dfhack.peristent.saveWorldData(key, data)`
- `dfhack.persistent.saveWorldDataString(key, data_str)`
- `dfhack.persistent.deleteWorldData(key)`

  Same semantics as for the `Site` functions, but will associated the data with the global world context.

The data is kept in memory, so no I/O occurs when getting or saving keys. It is all written to a json file in the game save directory when the game is saved.

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|item|plant|obj)`

  Uses type-specific methods for retrieving the code pair.

- `dfhack.matinfo.find(token[,token...])`

  Looks up material by a token string, or a pre-split string token sequence.

- `dfhack.matinfo.getToken(...)`, `info:getToken()`

  Applies `decode` and constructs a string token.

- `info:toString([temperature[,named]])`

  Returns the human-readable name at the given temperature.

- `info:getCraftClass()`

  Returns the classification used for craft skills.

- `info:matches(obj)`

  Checks if the material matches job_material_category or job_item. Accept dfhack_material_category auto-assign table.

Random number generation

- `dfhack.random.new([seed[,perturb_count]])`

  Creates a new random number generator object. Without any arguments, the object is initialized using current time. Otherwise, the seed must be either a non-negative integer, or a list of such integers. The second argument may specify the number of additional randomization steps performed to improve the initial state.

- `rng:init([seed[,perturb_count]])`

  Re-initializes an already existing random number generator object.

- `rng:random([limit])`

  Returns a random integer. If `limit` is specified, the value is in the range [0, limit); otherwise it uses the whole 32-bit unsigned integer range.

- `rng:drandom()`

  Returns a random floating-point number in the range [0,1).

- `rng:drandom0()`

  Returns a random floating-point number in the range (0,1).

- `rng:drandom1()`

  Returns a random floating-point number in the range [0,1].

- `rng:unitrandom()`

  Returns a random floating-point number in the range [-1,1].

- `rng:unitvector([size])`

  Returns multiple values that form a random vector of length 1, uniformly distributed over the corresponding sphere surface. The default size is 3.

- `fn = rng:perlin([dim]); fn(x[,y[,z]])`

  Returns a closure that computes a classical Perlin noise function of dimension *dim*, initialized from this random generator. Dimension may be 1, 2 or 3 (default).

C++ function wrappers

Thin wrappers around C++ functions, similar to the ones for virtual methods. One notable difference is that these explicit wrappers allow argument count adjustment according to the usual lua rules, so trailing false/nil arguments can be omitted.

- `dfhack.getOSType()`

  Returns the OS type string from `symbols.xml`.

- `dfhack.getDFVersion()`

  Returns the DF version string from `symbols.xml`.

- `dfhack.getDFHackVersion()`
- `dfhack.getDFHackRelease()`
- `dfhack.getDFHackBuildID()`
- `dfhack.getCompiledDFVersion()`
- `dfhack.getGitDescription()`
- `dfhack.getGitCommit()`
- `dfhack.getGitXmlCommit()`
- `dfhack.getGitXmlExpectedCommit()`
- `dfhack.gitXmlMatch()`
- `dfhack.isRelease()`
- `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.isSiteLoaded()`

  Checks if a site (e.g. a player fort) is 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.upperCp437(string)`

  Return a version of the string with all letters capitalized. Non-ASCII CP437 characters are capitalized if a CP437 version exists. For example, `ä` is replaced by `Ä`, but `â` is never capitalized.

- `dfhack.lowerCp437(string)`

  Return a version of the string with all letters in lower case. Non-ASCII CP437 characters are downcased. For example, `Ä` is replaced by `ä`.

- `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.capitalizeStringWords(string)`

  Return a version of the string with the first letter of each word capitalized. The beginning of a word is determined by a space or quote `"`. It is also determined by an apostrophe `'` when preceded by a space or comma. Non-ASCII CP437 characters will be capitalized if a CP437 version exists. This function does not downcase characters. Use `dfhack.lowerCp437` first, if desired.

- `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.getFocusStrings(viewscreen)`

  Returns a table of string representations of the current UI focuses. The strings have a "screen/foo/bar/baz..." format e.g..:

```
[1] = "dwarfmode/Info/CREATURES/CITIZEN"
[2] = "dwardmode/Squads"
```

- `dfhack.gui.matchFocusString(focus_string[, viewscreen])`

  Returns `true` if the given `focus_string` is found in the current focus strings, or as a prefix to any of the focus strings, or `false` if no match is found. Matching is case insensitive. If `viewscreen` is specified, gets the focus strings to match from the given viewscreen.

- `dfhack.gui.getCurFocus([skip_dismissed])`

  Returns the focus string of the current viewscreen.

- `dfhack.gui.getViewscreenByType(type[, depth])`

  Returns the topmost viewscreen out of the top `depth` viewscreens with the specified type (e.g. `df.viewscreen_titlest`), or `nil` if none match. If `depth` is not specified or is less than 1, all viewscreens are checked.

- `dfhack.gui.getDFViewscreen([skip_dismissed[, viewscreen]])`

  Returns the topmost viewscreen not owned by DFHack. If `skip_dismissed` is `true`, ignores screens already marked to be removed. If `viewscreen` is specified, starts the scan at the given viewscreen.

- `dfhack.gui.getWidget(container, <name or index>[, <name or index>...])`

  Returns the DF widget in the given widget container with the given name or (zero-based) numeric index. You can follow a chain of widget containers by passing additional names or indices. For example: `:lua ~dfhack.gui.getWidget(game.main_interface.info.labor, "Tabs", 0)`

- `dfhack.gui.getWidgetChildren(container)`

  Returns all the DF widgets in the given widget container.

General-purpose selections

- `dfhack.gui.getSelectedWorkshopJob([silent])`
- `dfhack.gui.getSelectedJob([silent])`
- `dfhack.gui.getSelectedUnit([silent])`
- `dfhack.gui.getSelectedItem([silent])`
- `dfhack.gui.getSelectedBuilding([silent])`
- `dfhack.gui.getSelectedCivZone([silent])`
- `dfhack.gui.getSelectedStockpile([silent])`

- `dfhack.gui.getSelectedPlant([silent])`

  Returns the currently selected in-game object or the indicated thing associated with the selected in-game object. For example, Calling `getSelectedJob` when a building is selected will return the job associated with the building (e.g. the `ConstructBuilding` job). If `silent` is ommitted or set to `false` and a selected object cannot be found, then an error is printed to the console.

- `dfhack.gui.getAnyWorkshopJob(screen)`
- `dfhack.gui.getAnyJob(screen)`
- `dfhack.gui.getAnyUnit(screen)`
- `dfhack.gui.getAnyItem(screen)`
- `dfhack.gui.getAnyBuilding(screen)`
- `dfhack.gui.getAnyCivZone(screen)`
- `dfhack.gui.getAnyStockpile(screen)`
- `dfhack.gui.getAnyPlant(screen)`

  Similar to the corresponding `getSelected` functions, but operate on the given screen instead of the current screen and always return `nil` silently on failure.

Fortress mode

- `dfhack.gui.getDwarfmodeViewDims()`

  Returns dimensions of the displayed map viewport. 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[,highlight]])` `dfhack.gui.revealInDwarfmodeMap(x,y,z[,center[,highlight]])`

  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. If `highlight` is true, then mark the target tile with a pulsing highlight until the player clicks somewhere else.

  `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 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[,update_alert])`

  Adds the report with the given index (returned by makeAnnouncement) to the specified group of the given unit. If `update_alert` is `true`, an alert badge will appear on the left side of the screen if not already visible. 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 based on the unit's current job and as requested by the flags. Always updates alert badges. Returns `true` on any success.

- `dfhack.gui.showAnnouncement(text[,color,is_bright])`

  Adds a regular announcement with given text, color, and brightness. The announcement type is always `df.announcement_type.REACHED_PEAK`, which uses the alert badge for `df.announcement_alert_type.GENERAL`.

- `dfhack.gui.showZoomAnnouncement(type,pos,text[,color,is_bright])`

  Like above, but also specifies a position you can zoom to from the announcement menu, as well as being able to set the announcement type.

- `dfhack.gui.showPopupAnnouncement(text[,color,is_bright])`

  Displays a megabeast-style modal announcement window. DF is currently ignoring the color and brightness settings (see: bug report.) Add `[C:` color `:0:` bright `]` (where color is 0-7 and bright is 0-1) in front of your text string to force the popup text to be colored.

  Text is run through a parser as it is converted into a markup text box. The parser accepts tokens in square brackets (`[ ]`.) Use `[[` and `]]` to include actual square brackets in text.

  The following tokens are accepted:

  `[R]`: (NEW_LINE) Ends the current line and begins on the next.

  `[B]`: (BLANK_LINE) Ends the current line and adds an additional blank line, beginning on the line after that.

  `[P]`: (INDENT) Ends the current line and begins four spaces indented on the next.

  `[CHAR:` n `]`, `[CHAR:~` ch `]`: Add a single character. First version takes a base-10 integer `n` representing a CP437 character. Second version accepts a character `ch` instead. `"[CHAR:154]"` and `"[CHAR:~"..string.char(154).."]"` both result in Ü. Use `[CHAR:32]` or `[CHAR:~ ]` to add extra spaces, which would normally be trimmed by the parser.

`[LPAGE:` link_type `:` id `]`, `[LPAGE:` link_type `:` id`` :`` subid `]`: Start a `markup_text_linkst`. These are intended for Legends mode page links and don't work in popups. The text will just be colored based on `link_type`. Valid link types are: `HF` (`HIST_FIG`,) `SITE`, `ARTIFACT`, `BOOK`, `SR` (`SUBREGION`,) `FL` (`FEATURE_LAYER`,) `ENT` (`ENTITY`,) `AB` (`ABSTRACT_BUILDING`,) `EPOP` (`ENTITY_POPULATION`,) `ART_IMAGE`, `ERA`, `HEC`. `subid` is only used for `AB` and `ART_IMAGE`. `[/LPAGE]` ends the link text.

`[C:` screenf `:` screenb `:` screenbright `]`: Color text. Sets the repective values in `df.global.gps` and then sets text color. `color` = `screenf`, `bright` = `screenbright`, `screenb` does nothing since popup backgrounds are always black. Example: `"Light gray, [C:4:0:0]red, [C:4:0:1]orange, [C:7:0:0]light gray."`

`[KEY:` n `]`: Keybinding. Shows the (first) keybinding for the `df.interface_key` `n`. The keybinding will be displayed in light green, but the previous text color will be restored afterwards.

- `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,unit_a,unit_d,is_sparring])`

  Takes a `df.announcement_infost` (see: structure definition) and a string and processes them just like DF does. Can also be built from parameters instead of an `announcement_infost`. 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.) Returns `true` if a report was created or repeated. For detailed info on why an announcement failed to display, enable `debugfilter set Debug core gui` in the DFHack console. If you want a guaranteed announcement, use `dfhack.gui.showAutoAnnouncement` instead.

- `dfhack.gui.getMousePos([allow_out_of_bounds])`

  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 a valid tile, returns `nil`. To allow the function to return coordinates outside of the map, set `allow_out_of_bounds` to `true`.

Other

- `dfhack.gui.getDepthAt(x, y)`

  Returns the distance from the z-level of the tile at map coordinates (x, y) to the closest rendered 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.removeJob(job)`

  Cancels a job, cleans up all references to it, and removes it from the world.

- `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.attachJobItem(job, item, role, filter_idx, insert_idx)`

  Attach a real item to this job. If the item is intended to satisfy a job_item filter, the index of that filter should be passed in `filter_idx`; otherwise, pass `-1`. Similarly, if you don't care where the item is inserted, pass `-1` for `insert_idx`. The `role` param is a `df.job_item_ref.T_role`. If the item needs to be brought to the job site, then the value should be `df.job_item_ref.T_role.Hauled`.

- `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[,include_insane])`

  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.isResident(unit[,include_insane])`

  The unit is a resident of the fortress. Same `include_insane` semantics as `isCitizen`.

- `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.isMarkedForTraining(unit)`
- `dfhack.units.isMarkedForTaming(unit)`
- `dfhack.units.isMarkedForWarTraining(unit)`
- `dfhack.units.isMarkedForHuntTraining(unit)`
- `dfhack.units.isMarkedForSlaughter(unit)`
- `dfhack.units.isMarkedForGelding(unit)`
- `dfhack.units.isGeldable(unit)`

- `dfhack.units.isGelded(unit)`
- `dfhack.units.isEggLayer(unit)`
- `dfhack.units.isEggLayerRace(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.isPet(unit)`
- `dfhack.units.isOpposedToLife(unit)`
- `dfhack.units.hasExtravision(unit)`
- `dfhack.units.isBloodsucker(unit)`

  Simple checks of caste attributes that can be modified by curses.
- `dfhack.units.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.isForgottenBeast(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, agitated wildlife, and crazed units.

- `dfhack.units.isGreatDanger(unit)`

  The unit is of Great Danger. This include demons, titans, forgotten beasts, 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.getUnitByNobleRole(role_name)`

  Returns the unit assigned to the given noble role, if any. `role_name` must be one of the position codes associated with the active fort or civilization government. For example: `CAPTAIN_OF_THE_GUARD`, `MAYOR`, or `BARON`. Note that if more than one unit has the role, only the first will be returned. See `getUnitsByNobleRole` below for retrieving all units with a particular role.

- `dfhack.units.getUnitsByNobleRole(role_name)`

  Returns a list of units (possibly empty) assigned to the given noble role.

- `dfhack.units.getCitizens([exclude_residents, [include_insane]])`

  Returns a list of all living, sane, citizens and residents that are currently on the map. Pass `exclude_residents` and `include_insane` both default to `false` but can be overridden.

- `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.assignTrainer(unit[, trainer_id])`
- `dfhack.units.unassignTrainer(unit)`

  Assignes (or unassigns) a trainer for the specified trainable unit. The trainer ID can be omitted if "any trainer" is desired. Returns a boolean indicating whether the operation was 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.casteFlagSet(race, caste, flag)`

  Returns whether the given `df.caste_raw_flags` flag is set for the given race and caste.

- `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.getReadableName(unit)`

  Returns a string that includes the language name of the unit (if any), the race of the unit, whether it is trained for war or hunting, any syndrome-given descriptions (such as "necromancer"), and the training level (if tame).

- `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.

Military module

- `dfhack.military.makeSquad(assignment_id)`

  Creates a new squad associated with the assignment (ie `df::entity_position_assignment`, via `id`) and returns it. Fails if a squad already exists that is associated with that assignment, or if the assignment is not a fort mode player controlled squad. Note: This function does not name the squad: consider setting a nickname (under `squad.name.nickname`), and/or filling out the `language_name` object at `squad.name`. The returned squad is otherwise complete and requires no more setup to work correctly.

- `dfhack.military.updateRoomAssignments(squad_id, assignment_id, squad_use_flags)`

  Sets the sleep, train, indiv_eq, and squad_eq flags when training at a barracks.

- `dfhack.military.getSquadName(squad_id)`

  Returns the name of a squad as a string.

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.getReadableDescription(item)`

  Returns a string generally fit to usefully describe the item to the player. When the item description appears anywhere in a script output or in the UI, this is usually the string you should use.

- `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[, caravan_state])`

  Calculates the value of an item. If a `df.caravan_state` object is given (from `df.global.plotinfo.caravans` or `df.global.main_interface.trade.mer`), then the value is modified by civ properties and any trade agreements that might be in effect.

- `dfhack.items.isRequestedTradeGood(item[, caravan_state])`

  Returns whether a caravan will pay extra for the given item. If caravan_state is not given, checks all active caravans.

- `dfhack.items.createItem(unit, item_type, item_subtype, mat_type, mat_index, growth_print, no_floor)`

  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)`

  Returns false if the item or any contained items cannot be traded.

- `canTradeAnyWithContents(item)`

  Returns true if the item is empty and can be traded or if the item contains any item that can be traded.

- `dfhack.items.markForTrade(item, depot)`

  Marks the given item for trade at the given depot.

- `dfhack.items.canMelt(item[, game_ui])`

  Returns true if the item can be designated for melting. Unless `game_ui` is given and true, bars, non-empty metal containers, and items in unit inventories are not considered meltable, even though they can be designated for melting using the game UI.

- `dfhack.items.markForMelting(item)`

  Marks the given item for melting, unless already marked. Returns true if the melting status was changed.

- `dfhack.items.cancelMelting(item)`

  Removes melting designation, if present, from the given item. Returns true if the melting status was changed.

- `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.

- `dfhack.items.getCapacity(item)`

  Returns the capacity volume of an item that can serve as a container for other items. Return value will be `0` for items that cannot serve as a container.

World module

- `dfhack.world.ReadPauseState()`

  Returns *true* if the game is paused.

- `dfhack.world.SetPauseState(paused)`

  Sets the pause state of the game.

- `dfhack.world.ReadCurrentYear()`

  Returns the current game year.

- `dfhack.world.ReadCurrentTick()`

  Returns the number of game ticks (`df.global.world.frame_counter`) since the start of the current game year.

- `dfhack.world.ReadCurrentMonth()`

  Returns the current game month, ranging from 0-11 (The Dwarven year has 12 months).

- `dfhack.world.ReadCurrentDay()`

  Returns the current game day, ranging from 1-28 (Each Dwarven month as 28 days)

- `dfhack.world.ReadCurrentWeather()`

  Returns the current game weather (`df.weather_type`).

- `dfhack.world.SetCurrentWeather(weather)`

  Sets the current game weather to `weather`.

- `dfhack.world.ReadWorldFolder()`

  Returns the name of the directory/folder the current saved game is under, or an empty string if no game was loaded this session.

- `dfhack.world.isFortressMode([gametype])`
- `dfhack.world.isAdventureMode([gametype])`
- `dfhack.world.isArena([gametype])`
- `dfhack.world.isLegends([gametype])`

  Without any arguments, returns *true* if the current gametype matches. Optionally accepts a gametype id to match against.

- `dfhack.world.getCurrentSite()`

  Returns the currently loaded `df.world_site` or `nil` if no site is loaded.

- `dfhack.world.getAdventurer()`

  Returns the current adventurer unit (if in adventure mode).

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.getBiomeType(region_coord2d)` or `getBiomeType(x,y)`

Returns the biome_type 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` and `getBiomeType`.

- `dfhack.maps.getPlantAtTile(pos)`, or `getPlantAtTile(x,y,z)`

  Returns the plant struct that owns the tile at the specified position.

- `dfhack.maps.getWalkableGroup(pos)`

  Returns the walkability group for the given tile position. A return value of `0` indicates that the tile is not walkable. The data comes from a pathfinding cache maintained by DF.

  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.canWalkBetween(pos1, pos2)`

  Checks if both positions are walkable and also share a walkability group.

- `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[, ignore_final_plus])`

  Returns the burrow pointer or *nil*. if `ignore_final_plus` is `true`, then + characters at the end of the names are ignored, both for the specified `name` and the names of the burrows that it matches against.

- `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(civzone,unit)`

  Replaces the owner of the civzone. 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,allow_flow])`

  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). Set `allow_flow` to true if the building can be built even if there is deep water or any magma on the tile (such as abstract buildings).

- `dfhack.buildings.countExtentTiles(extents,defval)`

  Returns the number of tiles included by extents, or defval.

- `dfhack.buildings.containsTile(building, x, y)`

  Checks if the building contains the specified tile. If the building contains extents, then the extents are checked. Otherwise, returns whether the x and y map coordinates are within the building's bounding box.

- `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.notifyCivzoneModified(building)`

  Rebuilds the civzone <-> overlapping building association mapping. Call after changing extents or modifying size in some fashion

- `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.

- `dfhack.buildings.completeBuild(building)`

  Complete an unconstructed or partially-constructed building and link it into the world.

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 UI interface tile the mouse is over, with the upper left corner being `0,0`. To get the map tile coordinate that the mouse is over, see `dfhack.gui.getMousePos()`.

- `dfhack.screen.getMousePixels()`

  Returns *x,y* of the screen coordinates the mouse is over in pixels, with the upper left corner being `0,0`.

- `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.

`keep_lower`
: If set to true, will not overwrite the background tile when filling in the foreground tile.

`write_to_lower`
: If set to true, the specified `tile` will be written to the background instead of the foreground.

`top_of_text`
: If set to true, the specified `tile` will have the top half of the specified `ch` character superimposed over the lower half of the tile.

`bottom_of_text`
: If set to true, the specified `tile` will have the bottom half of the specified `ch` character superimposed over the top half of the tile.

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 dialog where portions of the parent viewscreen are still visible in the background, call `screen:renderParent()`.

  If artifacts are left on the parent even after this function is called, such as when the window is dragged or is resized, any code can set `gui.Screen.request_full_screen_refresh` to `true`. Then when `screen.renderParent()` is next called, it will do a full flush of the graphics and clear the screen of artifacts.

- `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, _MOUSE_M`
  : If the left, right, and/or middle mouse button was just pressed.

  `_MOUSE_L_DOWN, _MOUSE_R_DOWN, _MOUSE_M_DOWN`
  : If the left, right, and/or middle mouse button is being held down.

  If this method is omitted, the screen is dismissed on reception of the `LEAVESCREEN` key.

- `function screen:onGetSelectedUnit()`
- `function screen:onGetSelectedItem()`
- `function screen:onGetSelectedJob()`
- `function screen:onGetSelectedBuilding()`
- `function screen:onGetSelectedStockpile()`
- `function screen:onGetSelectedCivZone()`
- `function screen:onGetSelectedPlant()`

  Override these if you want to provide a custom 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.

Textures module

In order for the game to render a particular tile (graphic), it needs to know the `texpos` - the position in the vector of the registered game textures (also the graphical tile id passed as the `tile` field in a *Pen*). Adding new textures to the vector is not difficult, but the game periodically deletes textures that are in the vector, and that's a problem since it invalidates the `texpos` value that used to point to that texture. The `textures` module solves this problem by providing a stable handle instead of a raw `texpos`. When we need to draw a particular tile, we can look up the current `texpos` value via the handle. Texture module can register textures in two ways: to reserved and dynamic ranges. Reserved range is a limit buffer in a game texture vector, that will never be wiped. It is good for static assets, which need to be loaded at the very beginning and will be used during the process running. In other cases, it is better to use dynamic range. If reserved range buffer limit has been reached, dynamic range will be used by default.

- `loadTileset(file, tile_px_w, tile_px_h[, reserved])`

  Loads a tileset from the image `file` with give tile dimensions in pixels. The image will be sliced in row major order. Returns an array of `TexposHandle`. `reserved` is optional boolean argument, which indicates texpos range. `true` - reserved, `false` - dynamic (default).

  Example usage:

```
local logo_textures = dfhack.textures.loadTileset('hack/data/art/dfhack.png', 8, 12)
local first_texposhandle = logo_textures[1]
```

- `getTexposByHandle(handle)`

  Get the current `texpos` for the given `TexposHandle`. Always use this method to get the `texpos` for your texture. `texpos` can change when game textures are reset, but the handle will be the same.

- `createTile(pixels, tile_px_w, tile_px_h[, reserved])`

  Create and register a new texture with the given tile dimensions and an array of `pixels` in row major order. Each pixel is an integer representing color in packed RBGA format (for example, #0022FF11). Returns a `TexposHandle`. `reserved` is optional boolean argument, which indicates texpos range. `true` - reserved, `false` - dynamic (default).

- `createTileset(pixels, texture_px_w, texture_px_h, tile_px_w, tile_px_h[, reserved])`

  Create and register a new texture with the given texture dimensions and an array of `pixels` in row major order. Then slice it into tiles with the given tile dimensions. Each pixel is an integer representing color in packed RBGA format (for example #0022FF11). Returns an array of `TexposHandle`. `reserved` is optional boolean argument, which indicates texpos range. `true` - reserved, `false` - dynamic (default).

- `deleteHandle(handle)`

  `handle` here can be single `TexposHandle` or an array of `TexposHandle`. Deletes all metadata and texture(s) related to the given handle(s). The handles become invalid after this call.

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.cxxDemangle(mangled_name)`

  Decodes a mangled C++ symbol name. Returns the demangled name on success, or `nil, error_message` on failure.

- `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. `dfhack-config/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.

- `dfhack.internal.msizeAddress(address)`

  Returns the allocation size of an address. Does not require a heap snapshot. This function will crash on an invalid pointer. Windows only.

- `dfhack.internal.getHeapState()`

  Returns the state of the heap. 0 == ok or empty, 1 == heap bad ptr, 2 == heap bad begin, 3 == heap bad node. Does not require a heap snapshot. This may be unsafe to use directly from lua if the heap is corrupt. Windows only.

- `dfhack.internal.heapTakeSnapshot()`

  Clears any existing heap snapshot, and takes an internal heap snapshot for later consumption. Windows only. Returns the same values as getHeapState()

- `dfhack.internal.isAddressInHeap(address)`

  Checks if an address is a member of the heap. It may be dangling. Requires a heap snapshot.

- `dfhack.internal.isAddressActiveInHeap(address)`

  Checks if an address is a member of the heap, and actively in use (ie valid). Requires a heap snapshot.

- `dfhack.internal.isAddressUsedAfterFreeInHeap(address)`

  Checks if an address is a member of the heap, but is not currently allocated (ie use after free). Requires a heap snapshot. Note that Windows eagerly removes freed pointers from the heap, so this is unlikely to trigger.

- `dfhack.internal.getAddressSizeInHeap(address)`

  Gets the allocated size of a member of the heap. Useful for detecting misaligns, as this does not return block size. Requires a heap snapshot.

- `dfhack.internal.getRootAddressOfHeapObject(address)`

  Gets the base heap allocation address of a address that lies internally within a piece of allocated memory. Eg, if you have a heap allocated struct and call this function on the address of the second member, it will return the address of the struct. Returns 0 if the address is not found. Requires a heap snapshot.

- `dfhack.internal.getClipboardTextCp437()`

  Gets the system clipboard text (and converts text to CP437 encoding).

- `dfhack.internal.setClipboardTextCp437(text)`

  Sets the system clipboard text from a CP437 string.

- `dfhack.internal.setClipboardTextCp437Multiline(text)`

  Sets the system clipboard text from a CP437 string. Character 0x10 is interpreted as a newline instead of the usual CP437 glyph.

- `dfhack.internal.getSuppressDuplicateKeyboardEvents()`
- `dfhack.internal.setSuppressDuplicateKeyboardEvents(suppress)`

  Gets and sets the flag for whether to suppress DF key events when a DFHack keybinding is matched and a command is launched.

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`.

7.6.3 Lua Modules

- *custom-raw-tokens*

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
  ```

  `COLOR_GREY` and `COLOR_DARKGREY` can also be spelled `COLOR_GRAY` and `COLOR_DARKGRAY`.

- 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.

- `ensure_keys(t, key...)`

  Walks a series of keys, creating any missing keys as empty tables. The new or existing table from the last specified key is returned from the function.

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.invert(table)`

  Returns a table where keys and values are reversed (i.e., a table containing a `value = key` entry for every `key = value` entry in the argument).

- `utils.tabulate(fun, start, stop[, step])`

  For numbers `start`, `stop` and `step`, with `step` defaulting to 1, returns a lua sequence `{ fun(start), fun(start+step), ... , fun(stop) }`.

- `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.search_text(text,search_tokens)`

  Returns true if all the search tokens are found within `text`. The text and search tokens are normalized to lower case and special characters (e.g. `A` with a circle on it) are converted to their "basic" forms (e.g. `a`). `search_tokens` can be a string or a table of strings. If it is a string, it is split into space-separated tokens before matching. The search tokens are treated literally, so any special regular expression characters do not need to be escaped. If `utils.FILTER_FULL_TEXT` is `true`, then the search tokens can match any part of `text`. If it is `false`, then the matches must happen at the beginning of words within `text`. You can change the value of `utils.FILTER_FULL_TEXT` in *gui/control-panel* on the "Preferences" tab.

- `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 arguments to options. Options and positional parameters can appear in any order on the commandline, as long as arguments to options immediately follow the option itself.

Each option specification in `optionActions` has the following format: `{shortOptionName, longOptionAlias, hasArg=boolean, handler=fn}`

- `shortOptionName` is a one-character string (or `''` or `nil` if the parameter only has a long form). Numbers cannot be short options, and negative numbers (e.g. `'-10'`) will be interpreted as positional parameters and returned in the positional parameters list.
- `longOptionAlias` is an optional longer form of the short option name. If no short option name is specified, then this element is required.
- `hasArg` indicates whether the handler function for the option takes a parameter.
- `handler` is the handler function for the option. If `hasArg` is `true` then the next token on the commandline is passed to the handler function as an argument.

Example usage:

```
local args = {...}
local open_readonly, filename = false, nil     -- set defaults

local positionals = argparse.processArgsGetopt(args, {
  {'r', handler=function() open_readonly = true end},
  {'f', 'filename', hasArg=true,
   handler=function(optarg) filename = optarg end}
  })
```

In this example, if `args` is `{'first', '-rf', 'fname', 'second'}` or, equivalently, `{'first', '-r', '--filename', 'myfile.txt', 'second'}` (note the double dash in front of the long option alias), then `open_readonly` will be `true`, `filename` will be `'myfile.txt'` and `positionals` will be `{'first', 'second'}`.

- `argparse.stringList(arg, arg_name, list_length)`

  Parses a comma-separated sequence of strings and returns a lua list. Leading and trailing spaces are trimmed from the strings. If `arg_name` is specified, it is used to make error messages more useful. If `list_length` is specified and greater than `0`, then exactly that number of elements must be found or the function will error. Example:

```
stringList('hello , world,alist', 'words') => {'hello', 'world', 'alist'}
```

- `argparse.numberList(arg, arg_name, list_length)`

  Parses a comma-separated sequence of numeric strings and returns a list of the discovered numbers (as numbers, not strings). If `arg_name` is specified, it is used to make error messages more useful. If `list_length` is specified and greater than `0`, exactly that number of elements must be found or the function will error. Example:

```
numberList('10, -20 ,  30.5') => {10, -20, 30.5}
```

- `argparse.coords(arg, arg_name, skip_validation)`

  Parses a comma-separated coordinate string and returns a coordinate table of `{x, y, z}`. If the string `'here'` is passed, returns the coordinates of the active game cursor, or throws an error if the cursor is not active. This function also verifies that the coordinates are valid for the current map and throws if they are not (unless `skip_validation` is set to true).

- `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.

- `argparse.boolean(arg, arg_name)`

  Converts `string.lower(arg)` from "yes/no/on/off/true/false/etc..." to a lua boolean. Throws if the value can't be converted, otherwise returns `true`/`false`. 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/tools`. 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 loaded when DFHack initializes, but can be explicitly refreshed with a call to `helpdb.refresh()` if docs are added/changed during a play session.

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.refresh()`

  Scan for changes in available commands and their documentation.
- `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 match succeeds if all 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:

```
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:

```
-- check projectile distance flown is zero, get firer, etc...
local multiplier = tonumber(customRawTokens.getToken(bow, "EXAMPLE_MOD_FIRE_RATE_
↪MULTIPLIER")) or 1
if firer.counters.think_counter > 0 then
  firer.counters.think_counter = math.max(math.floor(firer.counters.think_counter *␣
↪multiplier), 1)
end
```

- Something for a script that prints help text about different types of units:

```
local unit = dfhack.gui.getSelectedUnit()
if not unit then return end
local helpText = customRawTokens.getToken(unit, "EXAMPLE_MOD_HELP_TEXT")
if helpText then print(helpText) end
```

- Healing armour:

```
-- (per unit every tick)
local healAmount = 0
for _, entry in ipairs(unit.inventory) do
    if entry.mode == 2 then -- Worn
        healAmount = healAmount + tonumber((customRawTokens.getToken(entry.item,
↪"EXAMPLE_MOD_HEAL_AMOUNT")) or 0)
    end
end
```

(continues on next page)

(continued from previous page)

```
unit.body.blood_count = math.min(unit.body.blood_max, unit.body.blood_count +␣
↪healAmount)
```

7.6.4 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

- `CLEAR_PEN`

  The black pen used to clear the screen. In graphics mode, it will clear the foreground and set the background to the standard black tile.

- `TRANSPARENT_PEN`

  A pen that will clear all textures from the UI layer, making the tile transparent.

- `KEEP_LOWER_PEN`

  A pen that will write tiles over existing background tiles instead of clearing them.

- `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`.

  You can send mouse clicks as well by setting the `_MOUSE_L` key or other mouse-related pseudo-keys documented with the `screen:onInput(keys)` function above. Note that if you are simulating a click at a specific spot on the screen, you must set `df.global.gps.mouse_x` and `df.global.gps.mouse_y` if you are clicking on the interface layer or `df.global.gps.precise_mouse_x` and `df.global.gps.precise_mouse_y` if you are clicking on the map.

- `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.

- `invert_color(color, bold)`

  This inverts the brightness of `color`. If this color is coming from a pen's foreground color, include `pen.bold` in `bold` for this to work properly.

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. This can be a boolean or a function that returns a boolean.

active
: Specifies that the view should receive events, if also visible. This can be a boolean or a function that returns a boolean.

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:getMouseFramePos()`

  Returns the mouse *x,y* in coordinates local to `frame_rect` 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 modal 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.

ZScreen class

A screen subclass that allows multi-layer interactivity. For example, a DFHack GUI tool implemented as a ZScreen can allow the player to interact with the underlying map, or even other DFHack ZScreen windows! That is, even when the DFHack tool window is visible, players will be able to use vanilla designation tools, select units, and scan/drag the map around.

At most one ZScreen can have input focus at a time. That ZScreen's widgets will have a chance to handle the input before anything else. If unhandled, the input skips all unfocused ZScreens under that ZScreen and is passed directly to the first non-ZScreen viewscreen. There are class attributes that can be set to control what kind of unhandled input is passed to the lower layers.

If multiple ZScreens are visible and the player scrolls or left/right clicks on a visible element of a non-focused ZScreen, that ZScreen will be given focus. This allows multiple DFHack GUI tools to be usable at the same time. If the mouse is clicked away from the ZScreen widgets, that ZScreen loses focus. If no ZScreen has focus, all input is passed directly through to the first underlying non-ZScreen viewscreen.

For a ZScreen with keyboard focus, if `Esc` or the right mouse button is pressed, and the ZScreen widgets don't otherwise handle them, then the ZScreen is dismissed.

All this behavior is implemented in `ZScreen:onInput()`, which subclasses **must not override**. Instead, ZScreen subclasses should delegate all input processing to subviews. Consider using a *Window class* widget subview as your top level input processor.

When rendering, the parent viewscreen is automatically rendered first, so subclasses do not have to call `self:renderParent()`. Calls to `logic()` (a world "tick" when playing the game) are also passed through, so the game progresses normally and can be paused/unpaused as normal by the player. Note that passing `logic()` calls through to the underlying map is required for allowing the player to drag the map with the mouse. ZScreen subclasses can set attributes that control whether the game is paused when the ZScreen is shown and whether the game is forced to continue being paused while the ZScreen is shown. If pausing is forced, child `Window` widgets will show a force-pause indicator to show which tool is forcing the pausing.

ZScreen provides the following functions:

- `zscreen:raise()`

  Raises the ZScreen to the top of the viewscreen stack, gives it keyboard focus, and returns a reference to `self`. A common pattern is to check if a tool dialog is already active when the tool command is run and raise the existing dialog if it exists or show a new dialog if it doesn't. See the sample code below for an example.

- `zscreen:isMouseOver()`

  The default implementation iterates over the direct subviews of the ZScreen subclass (which usually only includes a single Window subview) and sees if `getMouseFramePos()` returns a position for any of them. Subclasses can override this function if that logic is not appropriate.

- `zscreen:hasFocus()`

  Whether the ZScreen has keyboard focus. Subclasses will generally not need to check this because they can assume if they are getting input, then they have focus.

ZScreen subclasses can set the following attributes:

- `defocusable` (default: `true`)

  Whether the ZScreen loses keyboard focus when the player clicks on an area of the screen other than the tool window. If the player clicks on a different ZScreen window, focus still transfers to that other ZScreen.

- `initial_pause` (default: `DEFAULT_INITIAL_PAUSE or not pass_mouse_clicks`)

  Whether to pause the game when the ZScreen is shown. If not explicitly set, this attribute will be true if the system-wide `DEFAULT_INITIAL_PAUSE` is `true` (which is its default value) or if the `pass_mouse_clicks` attribute is `false` (see below). It depends on `pass_mouse_clicks` because if the player normally pauses/unpauses the game with the mouse, they will not be able to pause the game like they usually do while the ZScreen has focus. `DEFAULT_INITIAL_PAUSE` can be customized permanently via *gui/control-panel* or set for the session by running a command like:

  ```
  :lua require('gui.widgets').DEFAULT_INITIAL_PAUSE = false
  ```

- `force_pause` (default: `false`)

  Whether to ensure the game *stays* paused while the ZScreen is shown, regardless of whether it has input focus.

- `pass_pause` (default: `true`)

  Whether to pass the pause key to the lower viewscreens if it is not handled by this ZScreen.

- `pass_movement_keys` (default: `false`)

  Whether to pass the map movement keys to the lower viewscreens if they are not handled by this ZScreen.

- `pass_mouse_clicks` (default: `true`)

  Whether to pass mouse clicks to the lower viewscreens if they are not handled by this ZScreen.

Here is an example skeleton for a ZScreen tool window:

```
local gui = require('gui')
local widgets = require('gui.widgets')

MyWindow = defclass(MyWindow, widgets.Window)
MyWindow.ATTRS {
    frame_title='My Window',
    frame={w=50, h=45},
    resizable=true, -- if resizing makes sense for your dialog
    resize_min={w=50, h=20}, -- try to allow users to shrink your windows
}

function MyWindow:init()
    self:addviews{
        -- add subview widgets here
    }
end

-- implement if you need to handle custom input
--function MyWindow:onInput(keys)
--    return MyWindow.super.onInput(self, keys)
--end

MyScreen = defclass(MyScreen, gui.ZScreen)
MyScreen.ATTRS {
    focus_path='myscreen',
```

(continues on next page)

(continued from previous page)

```
    -- set pause and passthrough attributes as appropriate
    -- (but most tools can use the defaults)
}

function MyScreen:init()
    self:addviews{MyWindow{}}
end

function MyScreen:onDismiss()
    view = nil
end

view = view and view:raise() or MyScreen{}:show()
```

ZScreenModal class

A ZScreen convenience subclass that sets the attributes to something appropriate for modal dialogs. The game is force paused, and no input is passed through to the underlying viewscreens.

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:

- `FRAME_WINDOW`

  A frame suitable for a draggable, optionally resizable window.

- `FRAME_PANEL`

  A frame suitable for a static (non-draggable, non-resizable) panel.

- `FRAME_MEDIUM`

  A frame suitable for overlay widget panels.

- `FRAME_THIN`

  A frame suitable for floating tooltip panels that need the DFHack signature.

- `FRAME_BOLD`

  A frame suitable for a non-draggable panel meant to capture the user's focus, like an important notification, confirmation dialog or error message.

- `FRAME_INTERIOR`

  A frame suitable for light interior accent elements. This frame does *not* have a visible `DFHack` signature on it, so it must not be used as the most external frame for a DFHack-owned UI.

- `FRAME_INTERIOR_MEDIUM`

  A copy of `FRAME_MEDIUM` that lacks the `DFHack` signature. Suitable for panels that are part of a larger widget cluster. Must *not* be used as the most external frame for a DFHack-owned UI.

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. Though this can be used for your "main window", see the *Window class* below for a more conveniently configured `Panel` subclass.

Has attributes:

- `subviews = {}`

  Used to initialize the subview list in the constructor.

- `on_render = function(painter)`

  Called from `onRenderBody`.

- `on_layout = function(frame_body)`

  Called from `postComputeFrame`.

- `draggable = bool` (default: `false`)
- `drag_anchors = {}` (default: `{title=true, frame=false/true, body=true}`)
- `drag_bound = 'frame' or 'body'` (default: `'frame'`)
- `on_drag_begin = function()` (default: `nil`)
- `on_drag_end = function(success, new_frame)` (default: `nil`)

  If `draggable` is set to `true`, then the above attributes come into play when the panel is dragged around the screen, either with the mouse or the keyboard. `drag_anchors` sets which parts of the panel can be clicked on with the left mouse button to start dragging. The frame is a drag anchor by default only if `resizable` (below) is `false`. `drag_bound` configures whether the frame of the panel (if any) can be dragged outside the containing parent's boundary. The body will never be draggable outside of the parent, but you can allow the frame to cross the boundary by setting `drag_bound` to `'body'`. The boolean passed to the `on_drag_end` callback will be

`true` if the drag was "successful" (i.e. not canceled) and `false` otherwise. Dragging can be canceled by right clicking while dragging with the mouse, hitting `Esc` (while dragging with the mouse or keyboard), or by calling `Panel:setKeyboaredDragEnabled(false)` (while dragging with the keyboard). If it is more convenient to do so, you can choose to override the `panel:onDragBegin` and/or the `panel:onDragEnd` methods instead of setting the `on_drag_begin` and/or `on_drag_end` attributes.

- `resizable = bool` (default: `false`)
- `resize_anchors = {}` (default: `{t=false, l=true, r=true, b=true}`
- `resize_min = {}` (default: w and h from the `frame`, or `{w=5, h=5}`)
- `on_resize_begin = function()` (default: `nil`)
- `on_resize_end = function(success, new_frame)` (default: `nil`)

  If `resizable` is set to `true`, then the player can click the mouse on any edge specified in `resize_anchors` and drag the border to resize the window. If two adjacent edges are enabled as anchors, then the tile where they meet can be used to resize both edges at the same time. The minimum dimensions specified in `resize_min` (or inherited from `frame` are respected when resizing. The panel is also prevented from resizing beyond the boundaries of its parent. When the player clicks on a valid anchor, `on_resize_begin()` is called. The boolean passed to the `on_resize_end` callback will be `true` if the drag was "successful" (i.e. not canceled) and `false` otherwise. Dragging can be canceled by right clicking while resizing with the mouse, hitting `Esc` (while resizing with the mouse or keyboard), or by calling `Panel:setKeyboardResizeEnabled(false)` (while resizing with the keyboard). If it is more convenient to do so, you can choose to override the `panel:onResizeBegin` and/or the `panel:onResizeEnd` methods instead of setting the `on_resize_begin` and/or `on_resize_end` attributes.

- `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`.

Has functions:

- `panel:setKeyboardDragEnabled(bool)`

  If called with `true` and the panel is not already in keyboard drag mode, then any current drag or resize operations are halted where they are (not canceled), the panel siezes input focus (see *View class* above for information on the DFHack focus subsystem), and further keyboard cursor keys move the window as if it were being dragged. Shift-cursor keys move by larger amounts. Hit `Enter` to commit the new window position or `Esc` to cancel. If dragging is canceled, then the window is moved back to its original position.

- `panel:setKeyboardResizeEnabled(bool)`

  If called with `true` and the panel is not already in keyboard resize mode, then any current drag or resize operations are halted where they are (not canceled), the panel siezes input focus (see *View class* above for information on the DFHack focus subsystem), and further keyboard cursor keys resize the window as if it were being dragged from the lower right corner. If neither the bottom or right edge is a valid anchor, an appropriate corner will be chosen. Shift-cursor keys move by larger amounts. Hit `Enter` to commit the new window size or `Esc` to cancel. If resizing is canceled, then the window size from before the resize operation is restored.

- `panel:onDragBegin()`

- `panel:onDragEnd(success, new_frame)`
- `panel:onResizeBegin()`
- `panel:onResizeEnd(success, new_frame)`

The default implementations of these methods call the associated attribute (if set). You can override them in a subclass if that is more convenient than setting the attributes.

Double clicking:

If the panel is resizable and the user double-clicks on the top edge (the frame title, if the panel has a frame), then the panel will jump to its maximum size. If the panel has already been maximized in this fashion, then it will jump to its minimum size. Both jumps respect the resizable edges defined by the `resize_anchors` attribute.

The time duration that a double click can span is defined by the global variable `DOUBLE_CLICK_MS`. The default value is `500` and can be changed by the end user with a command like:

```
:lua require('gui.widgets').DOUBLE_CLICK_MS=1000
```

Window class

Subclass of Panel; sets Panel attributes to useful defaults for a top-level framed, draggable window.

ResizingPanel class

Subclass of Panel; automatically adjusts its own frame height and width to the minimum required to show its subviews. Pairs nicely with a parent Panel that has `autoarrange_subviews` enabled.

It has the following attributes:

auto_height
: Sets self.frame.h from the positions and height of its subviews (default is `true`).

auto_width
: Sets self.frame.w from the positions and width of its subviews (default is `false`).

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.

Divider class

Subclass of Widget; implements a divider line that can optionally connect to existing frames via T-junction edges. A `Divider` instance is required to have a `frame` that is either 1 unit tall or 1 unit wide.

`Divider` widgets should be a sibling with the framed `Panel` that they are dividing, and they should be added to the common parent widget **after** the `Panel` so that the `Divider` can overwrite the `Panel` frame with the appropriate T-junction graphic. If the `Divider` will not have T-junction edges, then it could potentially be a child of the `Panel` since the `Divider` won't need to overwrite the `Panel`'s frame.

If two `Divider` widgets are set to cross, then you must have a third 1x1 `Divider` widget for the crossing tile so the other two `Divider`s can be seamlessly connected.

Attributes:

- `frame_style`

  The `gui` `FRAME` instance to use for the graphical tiles. Defaults to `gui.FRAME_THIN`.

- `interior`

  Whether the edge T-junction tiles should connect to interior lines (e.g. the vertical or horizontal segment of another `Divider` instance) or the exterior border of a `Panel` frame. Defaults to `false`, meaning exterior T-junctions will be chosen.

- `frame_style_t`
- `frame_style_b`
- `frame_style_l`
- `frame_style_r`

  Overrides for the frame style for specific T-junctions. Note that there are not currently any frame styles that allow borders of different weights to be seamlessly connected. If set to `false`, then the indicated edge will end in a straight segment instead of a T-junction.

- `interior_t`
- `interior_b`
- `interior_l`
- `interior_r`

  Overrides for the interior/exterior specification for specific T-junctions.

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)`.

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 (or r-clicking with the mouse) 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-B/Ctrl-F: move the cursor one word back or forward.
- Ctrl-A/Ctrl-E: move the cursor to the beginning/end of the text.

The widget also supports integration with the system clipboard:

- Ctrl-C: copy current text to the system clipboard
- Ctrl-X: copy current text to the system clipboard and clear text in widget
- Ctrl-V: paste text from the system clipboard (text is converted to cp437)

The `EditField` class also provides the following functions:

- `editfield:setCursor([cursor_pos])`

  Sets the text insert cursor to the specified position. If `cursor_pos` is not specified or is past the end of the current text string, the cursor will be set to the end of the current input (that is, `#editfield.text + 1`).

- `editfield:setText(text[, cursor_pos])`

Sets the input text string and, optionally, the cursor position. If the cursor position is not specified, it sets it to the end of the string.

- `editfield:insert(text)`

  Inserts the given text at the current cursor position.

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*). They are styled like scrollbars used in vanilla DF.

Scrollbars have the following attributes:

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.

If the mouse wheel is scrolled while the mouse is over the Scrollbar widget's parent view, then the parent is scrolled accordingly. Holding `Shift` while scrolling will result in faster movement.

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. By default, this will invert the foreground and background colors.

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.

`text_pen`, `text_dpen`, and `text_hpen` can either be a pen or a function that dynamically returns a pen.

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`, `token.htile`

  Specifies a pen or texture index (or a function that returns a pen or texture index) to paint as one tile before the main part of the token. If `htile` is specified, that is used instead of `tile` when the Label is hovered over with the mouse.

- `token.width = ...`

  If specified either as a value or a callback, the text (or tile) 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`, `token.hpen`

  Specify the pen, disabled pen, and hover pen to be used for the token's text. The fields 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`, `-halfpage`, `home`, or `end`. It returns the number of lines that were actually scrolled (negative for scrolling up).

- `label:shouldHover()`

  This method returns whether or not this widget should show a hover effect, generally you want to return `true` if there is some type of mouse handler present. For example, for a `HotKeyLabel`:

```
function HotkeyLabel:shouldHover()
    -- When on_activate is set, text should also hover on mouseover
    return HotkeyLabel.super.shouldHover(self) or self.on_activate
end
```

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.

The HotkeyLabel widget implements the following methods:

- `hotkeylabel:setLabel(label)`

  Updates the label without altering the hotkey text.

- `hotkeylabel:setOnActivate(on_activate)`

  Updates the on_activate callback.

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'`.

key_back
: Similar to `key`, but will cycle backwards (optional)

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.

label_width
: The number of spaces to allocate to the `label` (for use in aligning a column of `CycleHotkeyLabel` labels).

label_below
: If `true`, then the option value will appear below the label instead of to the right of it. Defaults to `false`.

option_gap
: The size of the gap between the label text and the option value. Default is `1`. If set to `0`, there'll be no gap between the strings. If `label_below == true`, negative values will shift the value leftwards.

options
: A list of strings or tables of `{label=string or fn, value=val[, pen=pen]}`. String options use the same string for the label and value and use the default pen. The optional `pen` element could be a color like `COLOR_RED`.

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([backwards])`

  Cycles the selected option and triggers the `on_change` callback. If `backwards` is defined and is truthy, the cycle direction will be reversed

- `cyclehotkeylabel:setOption(value_or_index, call_on_change)`

  Sets the current option to the option with the specified value or index. If `call_on_change` is set to `true`, then the `on_change` callback is triggered.

- `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.

- `cyclehotkeylabel:getOptionPen([option_idx])`

  Retrieves the option pen at the given index, or the pen of the currently selected option if no index is given. If an option was defined as just a string, then this function will return `nil` for that option.

ToggleHotkeyLabel class

This is a specialized subclass of CycleHotkeyLabel that has two options: `On` (with a value of `true`) and `Off` (with a value of `false`). The `On` option is rendered in green.

HelpButton class

A 3x1 tile button with a question mark on it, intended to represent a help icon. Clicking on the icon will launch *gui/launcher* with a given command string, showing the help text for that command.

It has the following attributes:

command
: The command to load in *gui/launcher*.

It also sets the `frame` attribute so the button appears in the upper right corner of the parent, but you can override this to your liking if you want a different position.

ConfigureButton class

A 3x1 tile button with a gear mark on it, intended to represent a configure icon. Clicking on the icon will run the given callback.

It has the following attributes:

on_click
: The function on run when the icon is clicked.

BannerPanel class

This is a Panel subclass that prints a distinctive banner along the far left and right columns of the widget frame. Note that this is not a "proper" frame since it doesn't have top or bottom borders. Subviews of this panel should inset their frames one tile from the left and right edges.

TextButton class

This is a BannerPanel subclass that wraps a HotkeyLabel with some decorators on the sides to make it look more like a button, suitable for both graphics and ASCII modes. All HotkeyLabel parameters passed to the constructor are passed through to the wrapped HotkeyLabel.

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.

text_hpen
: Specifies the pen for entries that the mouse is hovered over. Defaults to swapping the background/foreground colors.

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-click callback; if specified, the list reacts to the click and calls the callback as `on_submit2(index,choice)`.

on_double_click
: Mouse double click callback; if specified, the list reacts to the click and calls the callback as `on_double_click(index,choice)`.

on_double_click2
: Shift-double click callback; if specified, the list reacts to the click and calls the callback as `on_double_click2(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.

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_change
: If specified, will be passed to the filter edit field as its `on_change` 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.

Filter behavior:

By default, the filter matches substrings that start at the beginning of a word (or after any punctuation). You can instead configure filters to match any substring across the full text with a command like:

```
:lua require('utils').FILTER_FULL_TEXT=true
```

TabBar class

This widget implements a set of one or more tabs to allow navigation between groups of content. Tabs automatically wrap on the width of the window and will continue rendering on the next line(s) if all tabs cannot fit on a single line.

key
Specifies a keybinding that can be used to switch to the next tab. Defaults to `CUSTOM_CTRL_T`.

key_back
Specifies a keybinding that can be used to switch to the previous tab. Defaults to `CUSTOM_CTRL_Y`.

labels
A table of strings; entry representing the label text for a single tab. The order of the entries determines the order the tabs will appear in.

on_select
Callback executed when a tab is selected. It receives the selected tab index as an argument. The provided function should update the value of whichever variable your script uses to keep track of the currently selected tab.

get_cur_page
The function used by the TabBar to determine which Tab is currently selected. The function you provide should return an integer that corresponds to the non-zero index of the currently selected Tab (i.e. whatever variable you update in your `on_select` callback)

active_tab_pens
A table of pens used to render active tabs. See the default implementation in widgets.lua for an example of how to construct the table. Leave unspecified to use the default pens.

inactive_tab_pens
A table of pens used to render inactive tabs. See the default implementation in widgets.lua for an example of how to construct the table. Leave unspecified to use the default pens.

get_pens
A function used to determine which pens should be used to render a tab. Receives the index of the tab as the first argument and the TabBar widget itself as the second. The default implementation, which will handle most situations, returns `self.active_tab_pens`, if `self.get_cur_page() == idx`, otherwise returns `self.inactive_tab_pens`.

Tab class

This widget implements a single clickable tab and is the main component of the TabBar widget. Usage of the `TabBar` widget does not require direct usage of `Tab`.

id
The id of the tab.

label
The text displayed on the tab.

on_select
Callback executed when the tab is selected.

get_pens
A function that is used during `Tab:onRenderBody` to determine the pens that should be used

for drawing. See the usage of `Tab` in `TabBar:init()` for an example. See the default value of `active_tab_pens` or `inactive_tab_pens` in `TabBar` for an example of how to construct pens.

RangeSlider class

This widget implements a mouse-interactable range-slider. The player can move its two handles to set minimum and maximum values to define a range, or they can drag the bar itself to move both handles at once. The parent widget owns the range values, and can control them independently (e.g. with `CycleHotkeyLabels`). If the range values change, the `RangeSlider` appearance will adjust automatically.

num_stops
Used to specify the number of "notches" in the range slider, the places where handles can stop. (this should match the parents' number of options)

get_left_idx_fn
The function used by the RangeSlider to get the notch index on which to display the left handle.

get_right_idx_fn
The function used by the RangeSlider to get the notch index on which to display the right handle.

on_left_change
Callback executed when moving the left handle.

on_right_change
Callback executed when moving the right handle.

gui.textures

This module contains convenience methods for accessing default DFHack graphic assets. Pass the `offset` in tiles (in row major position) to get a particular tile from the asset. `offset` 0 is the first tile.

- `tp_green_pin(offset)` tileset: `hack/data/art/green-pin.png`
- `tp_red_pin(offset)` tileset: `hack/data/art/red-pin.png`
- `tp_icons(offset)` tileset: `hack/data/art/icons.png`
- `tp_on_off(offset)` tileset: `hack/data/art/on-off.png`
- `tp_control_panel(offset)` tileset: `hack/data/art/control-panel.png`
- `tp_border_thin(offset)` tileset: `hack/data/art/border-thin.png`
- `tp_border_medium(offset)` tileset: `hack/data/art/border-medium.png`
- `tp_border_bold(offset)` tileset: `hack/data/art/border-bold.png`
- `tp_border_panel(offset)` tileset: `hack/data/art/border-panel.png`
- `tp_border_window(offset)` tileset: `hack/data/art/order-window.png`

Example usage:

```
local textures = require('gui.textures')
local first_border_texpos = textures.tp_border_thin(1)
```

7.6.5 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
    {{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.

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 randomization state is kept in an "engine". The distribution class turns that engine state into random numbers.

Example:

```
local rng = require('plugins.cxxrandom')
local norm_dist = rng.normal_distribution(6820, 116)   -- avg, stddev
local engID = rng.MakeNewEngine(0)
print(norm_dist:next(engID))

-- alternate syntax
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 engID = rng.MakeNewEngine(0)
local g = rng.crng:new(engId, 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.

tiletypes

- `bool tiletypes_setTile(pos, shape, material, special, variant)` where the parameters are enum values from `df.tiletype_shape`, `df.tiletype_material`, etc. Returns whether the conversion succeeded.

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(
```

(continues on next page)

(continued from previous page)

```
        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
)
```

7.6.6 Scripts

Any files with the `.lua` extension placed into the `hack/scripts` folder (or any other folder in your *Script paths*) 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:

- `dfhack-config/scripts/startup.lua` is invoked as `startup`
- `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 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 your *Script paths*, 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 (i.e. built-in commands or commands provided by plugins), see `dfhack.run_command()`. Note that this is slightly slower than `dfhack.run_script()` when running 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 near the top of their file:

```
--@enable = true
--@module = true
```

Note that enableable scripts must also be *modules* so their `isEnabled()` functions can be called from outside the script.

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

If you declare a global function named `isEnabled` that returns a boolean indicating whether your script is enabled, then your script will be listed among the other enableable scripts and plugins when the player runs the *enable* command.

Example usage:

```
--@enable = true
--@module = true

enabled = enabled or false
function isEnabled()
    return enabled
end

-- (function definitions...)

if dfhack_flags.enable then
    if dfhack_flags.enable_state then
        start()
        enabled = true
    else
        stop()
        enabled = false
    end
end
```

If the state of your script can be tied to an active savegame, then your script should hook the appropriate events to load persisted state when a savegame is loaded. For example:

```
local json = require('json')
local persist = require('persist-table')

local GLOBAL_KEY = 'my-script-name'
g_state = g_state or {}

dfhack.onStateChange[GLOBAL_KEY] = function(sc)
    if sc ~= SC_MAP_LOADED or df.global.gamemode ~= df.game_mode.DWARF then
        return
    end
    local state = json.decode(persist.GlobalTable[GLOBAL_KEY] or '')
    g_state = state or {}
end
```

The attachment to `dfhack.onStateChange` should appear in your script code outside of any function. DFHack will load your script as a module just before the `SC_DFHACK_INITIALIZED` state change event is sent, giving your code an opportunity to run and attach hooks before the game is loaded.

If an enableable script is added to a DFHack *script path* while DF is running, then it will miss the initial sweep that loads all the module scripts and any `onStateChange` handlers the script may want to register will not be registered until the script is loaded via some means, either by running it or loading it as a module. If you just added new scripts that you want to load so they can attach their `onStateChange` handlers, run `enable` without parameters or call `:lua require('script-manager').reload()` to scan and load all script modules.

Save init script

If a save directory contains a file called `init.lua`, it is automatically loaded and executed every time the save is loaded. The same applies to any files called `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`.

7.7 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.

7.7.1 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 or lifecycle management code that would normally be required for you to show info in the UI.

7.7.2 Overlay widget API

Overlay widgets are Lua classes that inherit from `overlay.OverlayWidget` (which itself inherits from *widgets.Panel*). 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 *widgets.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. See the *hotkeys* DFHack logo widget for an example.

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()` (or just leave them at the defaults).

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. You should not set this property yourself.
- **`version`**
  You can set this to any string. If the version string of a loaded widget does not match the saved settings for that widget, then the configuration for the widget (position, enabled status) will be reset to defaults.
- **`desc`**
  A short (<100 character) description of what the overlay does. This text will be displayed in *gui/control-panel* on the "Overlays" tab.
- **`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.
- **`default_enabled` (default: `false`)**
  Override this attribute if the overlay should be enabled by default if it does not already have a state stored in `dfhack-config/overlay.json`.
- **`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. If you only want your widget to appear in certain contexts, you can specify a focus path, in the same syntax as the *keybinding* command. For example, `dwarfmode/Info/CREATURES/CITIZEN` will ensure the overlay widget is only displayed when the "Citizens" subtab under the "Units" panel is active.
- **`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. If you change the value of this attribute dynamically, it may not be noticed until the previous timeout expires. However, if you need a burst of high-frequency updates, set it to `0` and it will be noticed immediately.

Common widget attributes such as `active` and `visible` are also respected.

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. Plugin lua code is loaded with `require()` and script lua code is loaded with `reqscript()`. If your widget is in a script, ensure your script can be *loaded as a module*, or else the widget will not be discoverable. Whether the widget is enabled 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.

7.7.3 Development workflows

When you are developing an overlay widget, you will likely need to reload your widget many times as you make changes. The process for this differs slightly depending on whether your widget is attached to a plugin or is implemented in a script.

Note that reloading a script does not clear its global environment. This is fine if you are changing existing functions or adding new ones. If you remove a global function or other variable from the source, though, it will stick around in your script's global environment until you restart DF or run *devel/clear-script-env*.

Scripts

1. Edit the widget source
2. If the script is not in your *Script paths*, install your script (see the *DFHack modding guide* for help setting up a dev environment so that you don't need to reinstall your scripts after every edit).
3. Call `:lua require('plugins.overlay').rescan()` to reload your overlay widget

Plugins

1. Edit the widget source
2. Install the plugin so that the updated code is available in `hack/lua/plugins/`
3. If you have changed the compiled plugin, *reload* it
4. If you have changed the lua code, run `:lua reload('plugins.mypluginname')`
5. Call `:lua require('plugins.overlay').rescan()` to reload your overlay widget

7.7.4 Troubleshooting

You can check that your widget is getting discovered by the overlay framework by running `overlay list` or by launching *gui/control-panel* and checking the `Overlays` tab.

If your widget is not listed, double check that:

1. `OVERLAY_WIDGETS` is declared, is global (not `local`), and references your widget class
2. (if a script) your script is *declared as a module* (`--@ module = true`) and it does not have side effects when loaded as a module (i.e. you check `dfhack_flags.module` and return before executing any statements if the value is `true`)
3. your code does not have syntax errors – run `:lua ~reqscript('myscriptname')` (if a script) or `:lua ~require('plugins.mypluginname')` (if a plugin) and make sure there are no errors and the global environment contains what you expect.

If your widget is not running when you expect it to be running, run *gui/overlay* when on the target screen and check to see if your widget is listed when showing overlays for the current screen. If it's not there, verify that this screen is included in the `viewscreens` list in the widget class attributes. Also, load *gui/control-panel* and make sure your widget is enabled.

7.7.5 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{
    desc='Sample widget that displays a message on the screen.',
    default_pos={x=5,y=-2},
    default_enabled=true,
    viewscreens={'dwarfmode', 'dungeonmode'},
    overlay_onupdate_max_freq_seconds=20,
}

function MessageWidget:init()
    self:addviews{
        widgets.Label{
            view_id='label',
            text='',
        },
    }
end

function MessageWidget:overlay_onupdate()
    local text = getImportantMessage() -- defined in the host script/plugin
    self.subviews.label:setText(text)
    self.frame.w = #text
end

function MessageWidget:onInput(keys)
```

(continues on next page)

(continued from previous page)

```
    if keys.CUSTOM_ALT_Z then
        self:overlay_onupdate()
        return true
    end
    return MessageWidget.super.onInput(self, keys)
end

OVERLAY_WIDGETS = {message=MessageWidget}
```

7.7.6 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{
    desc='Sample widget that highlights artifacts on the game map.',
    default_enabled=true,
    viewscreens={'dwarfmode', 'dungeonmode'},
    overlay_onupdate_max_freq_seconds=10,
}

function ArtifactRadarWidget:overlay_onupdate()
    self.visible_artifacts_coords = getVisibleArtifactCoords()
end

function ArtifactRadarWidget:onRenderFrame()
    for _,pos in ipairs(self.visible_artifacts_coords) do
        -- highlight tile at given coordinates
    end
end

OVERLAY_WIDGETS = {radar=ArtifactRadarWidget}
```

7.7.7 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{
    desc='Sample widget that reacts to mouse hover.',
    default_pos={x=-3,y=-3},
```

(continues on next page)

(continued from previous page)

```
    default_enabled=true,
    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.ZScreen)
MenuScreen.ATTRS{
    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.Window{
            frame=frame,
            autoarrange_subviews=true,
            subviews={
                -- ...
                },
            },
        },
```

(continues on next page)

(continued from previous page)

```
    }
end
```

7.8 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.

Please see the following page for detailed information about the syntax of the df-structures XML files:

7.8.1 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...'/>
    ...
```

(continues on next page)

(continued from previous page)

```
    <global-address name='...' [value='0x...']/>
    ...

    <vtable-address name='...' [value='0x...'] [base='...']/>
    ...
</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. If a `base` attribute is specified, it must be the name of a loaded library. That is, the filename without any path components. The `value` will then be interpreted as an offset from the load address of that library.

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.

7.9 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

7.9.1 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.

7.9.2 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.

7.9.3 Windows-specific tools

Some people have used Cheat Engine for research in the past.

7.10 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

7.10.1 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.

7.10.2 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.

7.10.3 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.

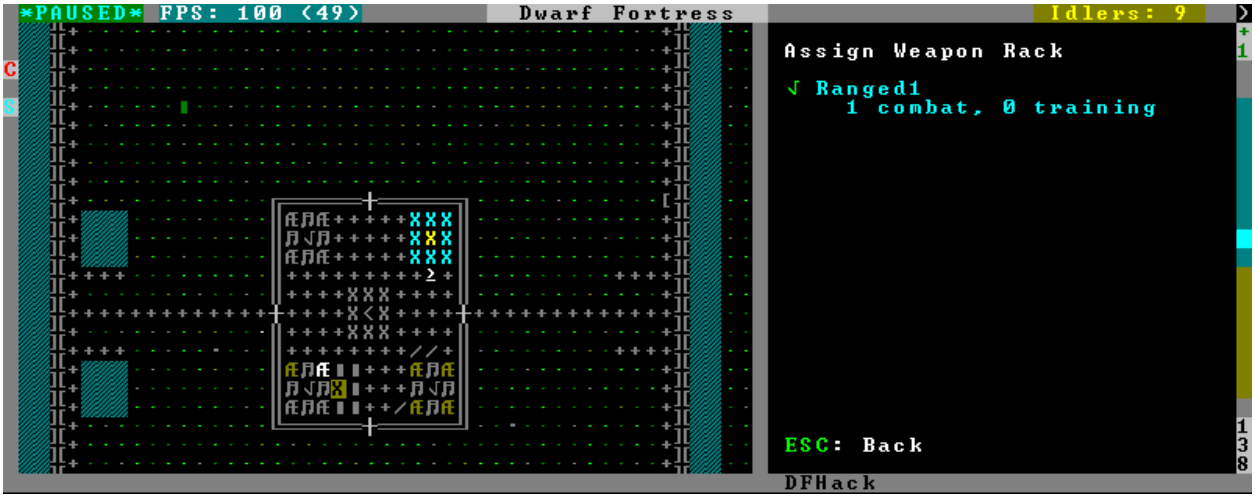
Contrary to the common misconception, all these uses are controlled by the *Individual Equipment* usage flag. The *Squad Equipment* flag is actually intended for ammo, but the game does even less in that area than for armor and weapons. This plugin implements the following rules almost from scratch:

- 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.



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.

7.11 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

    * *header*
    * *request*
    * *text*
    * *result*
    * *failure*
    * *quit*

7.11.1 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.

7.11.2 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

7.11.3 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)`

7.12 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

- *DFHack 0.44.02-beta1*
- *DFHack 0.44.02-alpha1*

7.12.1 DFHack 50.13-r2

New Tools

- **Updated for adventure mode:**
  - *reveal*
  - *gui/sandbox*, *gui/create-item*, *gui/reveal*
- *adaptation*: (reinstated) inspect or set unit cave adaptation levels
- *fix/engravings*: fix corrupt engraving tiles
- *flashstep*: (reinstated) teleport your adventurer to the mouse cursor
- *ghostly*: (reinstated) allow your adventurer to phase through walls
- *markdown*: (reinstated) export description of selected unit or item to a text file
- *resurrect-adv*: (reinstated) allow your adventurer to recover from death
- *reveal-adv-map*: (reinstated) reveal (or hide) the adventure map
- *unretire-anyone*: (reinstated) choose anybody in the world as an adventurer

New Features

- DFHack and the Dwarf Fortress translation project can now both be run at the same time
- *buildingplan*: quick material filter favorites on main planner panel
- *instruments*: new subcommand `instruments order` for creating instrument work orders

Fixes

- *blueprint*: correctly define stockpile boundaries in recorded stockpile ("place") blueprints when there are adjacent non-rectangular stockpiles of identical types
- *caravan*: don't include undiscovered divine artifacts in the goods list
- *combine*: respect container volume limits
- ***dig*:**
  - refresh count of tiles that will be modified by "mark all designated tiles on this z-level for warm/damp dig" when the z-level changes
  - don't affect already-revealed tiles when marking z-level for warm/damp dig
- *gui/quantum*: fix processing when creating a quantum dump instead of a quantum stockpile
- *logistics*: include semi-wild pets when autoretrain is enabled
- *modtools/create-item*: now functions properly when the `reaction-gloves` tweak is active
- *prospector*: don't use scientific notation for representing large numbers

- ***quickfort***:
  - don't designate multiple tiles of the same tree for chopping when applying a tree chopping blueprint to a multi-tile tree
  - fix detection of valid tiles for wells
- *suspendmanager*: fully suspend unbuildable dead ends (e.g. buildling second level of a wall when the wall top is only accessible via ramp, causing the planned wall to be pathable but not buildable)
- ***zone***:
  - fix display of distance from cage/pit for small pets in assignment dialog
  - refresh values in distance column when switching selected pastures when the assign animals dialog is open

Misc Improvements

- Dreamfort: move wells on services level so brawling drunken tavern patrons are less likely to fall in
- New commandline options for controlling the Cloud Save coprocess when launching from Steam. See the *DFHack Core* documentation for details.
- *caravan*: display who is in the cages you are selecting for trade and whether they are hostile
- *combine*: reduce combined drink sizes to 25
- *deathcause*: automatically find and choose a corpse when a pile of mixed items is selected
- ***dig***:
  - warm/damp/aquifer status will now be shown in mining mode for tiles that your dwarves can see from the level below
  - warm/damp/aquifer status will now be shown when in smoothing/engraving modes
- *flashstep*: new keybinding for teleporting adventurer to the mouse cursor: Ctrl-t (when adventure map is in the default state and mortal mode is disabled in DFHack preferences)
- *gui/autobutcher*: add shortcuts for butchering/unbutchering all animals
- *gui/launcher*: add button for copying output to the system clipboard
- ***gui/quantum***:
  - add option for whether a minecart automatically gets ordered and/or attached
  - when attaching a minecart, show which minecart was attached
  - allow multiple feeder stockpiles to be linked to the minecart route
- *markdown*: new keybinding for triggering text export: Ctrl-t (when unit or item is selected)
- *prioritize*: add PutItemOnDisplay jobs to the default prioritization list – when these kinds of jobs are requested by the player, they generally want them done ASAP
- ***regrass***:
  - can now add grass to stairs, ramps, ashes, buildings, muddy stone, shrubs, and trees
  - can now restrict area of effect to specified tile, block, cuboid, or z-levels
  - can now add grass in map blocks where there hasn't been any
  - can now choose specific grass type

- *stockpiles*: support import and export "desired items" configuration for route stops
- *unretire-anyone*: new keybinding for adding a historical figure to the adventurer selection list in the adventure mode setup screen: Ctrl-a

API

- `dfhack.items.getReadableDescription()`: easy API for getting a human-readable item description with useful annotations and information (like tattered markers or who is in a cage)
- `Items::createItem`: now returns a list of item pointers rather than a single ID, moved creator parameter to beginning, added growth_print and no_floor parameters at end
- `World::getAdventurer`: returns current adventurer unit
- `World::ReadPauseState`: now returns true when the game is effectively paused due to a large panel obscuring the map. this aligns the return value with the visual state of the pause button when in fort mode.

Lua

- `dfhack.internal.setClipboardTextCp437Multiline`: for copying multiline text to the system clipboard
- `dfhack.items.createItem`: return value and parameters have changed as per C++ API
- `dfhack.world.getAdventurer`: returns current adventurer unit

Documentation

- Quickfort Blueprint Library: add demo videos for pump stack and light aquifer tap blueprints
- Update docs for dependency requirements and compilation procedures

7.12.2 DFHack 50.13-r1.1

Fixes

- *deathcause*: fix error on run
- *gui/quantum*: accept all item types in the output stockpile as intended

Documentation

- Update docs on release procedures and symbol generation

7.12.3 DFHack 50.13-r1

New Tools

- *gui/quantum*: (reinstated) point and click interface for creating quantum stockpiles or quantum dumps
- *gui/unit-info-viewer*: (reinstated) give detailed information on a unit, such as egg laying behavior, body size, birth date, age, and information about their afterlife behavior (if a ghost)

Fixes

- Fixed incorrect DFHack background window texture when DF is started in ascii mode and subsequently switched to graphics mode
- Fixed misidentification of visitors from your own civ as residents; affects all tools that iterate through citizens/residents
- *cursecheck*: act on selected unit only if a unit is selected
- *exterminate*: don't classify dangerous non-invader units as friendly (e.g. snatchers)
- ***gui/create-item*:**
  - properly restrict bags to bag materials by default
  - allow gloves and shoees to be made out of textiles by default
- *open-legends*: don't interfere with the dragging of vanilla list scrollbars

Misc Improvements

- *gui/gm-unit*: changes to unit appearance will now immediately be reflected in the unit portrait
- *open-legends*: allow player to cancel the "DF will now exit" dialog and continue browsing
- *suspendmanager*: Account for walls planned on the z-layer below when determining accessibility to a job

Structures

- `biome_type`: add enum attrs for `caption` and `plant_raw_flags`

Documentation

- *autoclothing*: add section comparing `autoclothing` and *tailor* to guide players choosing which to enable

7.12.4 DFHack 50.12-r3

New Tools

- *aquifer*: commandline tool for creating, draining, and modifying aquifers
- *gui/aquifer*: interactive aquifer visualization and editing
- *open-legends*: (reinstated) open legends mode directly from a loaded fort

New Features

- ***blueprint*:**
  - designations and active dig jobs are now captured in generated blueprints
  - warm/damp dig markers are captured in generated blueprints
- *buildingplan*: add overlays for unlinking and freeing mechanisms from buildings
- ***dig*:**
  - designate tiles for damp or warm dig, which allows you to dig through damp or warm tiles without designations being canceled
  - damp and warm tile icons now remain visible when included in the designation selection box (graphics mode)
  - aquifer tiles are now visually distinct from "just damp" tiles (graphics and ascii modes)
  - light aquifer tiles are now visually distinct from heavy aquifer tiles (graphics and ascii modes)
  - autodig designations that are marked for damp/warm dig propagate the damp/warm tag when expanding to newly exposed tiles
- *gui/notify*: optional notification for general wildlife (not on by default)
- *gui/quickfort*: add options for setting warm/damp dig markers when applying blueprints
- *gui/reveal*: new "aquifer only" mode to only see hidden aquifers but not reveal any tiles
- *quickfort*: add options for setting warm/damp dig markers when applying blueprints

Fixes

- fix behavior of Linux Steam launcher on systems that don't support the inotify API
- fix rendering of resize "notch" in lower right corner of resizable windows in ascii mode
- *agitation-rebalance*: fix calculated percent chance of cavern invasion
- *armoks-blessing*: fix error when making "Normal" attributes legendary
- *emigration*: remove units from burrows when they emigrate
- *fix/loyaltycascade*: fix edge case where loyalties of renegade units were not being fixed
- *gui/launcher*: don't pop up a result dialog if a command run from minimal mode has no output
- ***quickfort*:**
  - stockpiles can now be placed even if there is water covering the tile, as per vanilla behavior
  - reject tiles for building that contain magma or deep water
- *stonesense*: fix a crash with buildings made of unusual materials (such as campsite tents made out of organic "walls")
- *suspendmanager*: prevent cancellation spam when an item is preventing a building from being completed

Misc Improvements

- aquifer_tap blueprint: now designates in damp dig mode for uninterrupted digging in a light aquifer
- pump_stack blueprint: now designates in warm and damp dig mode for uninterrupted digging through warm and damp tiles
- *agitation-rebalance*: when more than the maximum allowed cavern invaders are trying to enter the map, prefer keeping the animal people invaders instead of their war animals
- *gui/control-panel*: add alternate "nodump" version for *cleanowned* that does not cause citizens to toss their old clothes in the dump. this is useful for players who would rather sell old clothes than incinerate them
- *gui/reveal*: show aquifers even when not in mining mode
- *keybinding*: you can now assign keybindings to mouse buttons (if your mouse has more than the three buttons already used by DF)
- *tailor*: allow turning off automatic confiscation of tattered clothing

Removed

- *drain-aquifer*: replaced by `aquifer drain --all`; an alias now exists so `drain-aquifer` will automatically run the new command

API

- `Buildings::checkFreeTiles`: now takes a `allow_flow` parameter to control whether water- or magma-filled tiles are valid
- `Units::citizensRange`: c++-20 std::range filter for citizen units
- `Units::forCitizens`: iterator callback function for citizen units
- `Units::paintTile`, `Units::readTile`: now takes an optional field specification for reading and writing to specific map compositing layers

Lua

- `dfhack.gui.matchFocusString`: focus string matching is now case sensitive (for performance reasons)

Structures

- name many previously-unknown map-related fields and flag bits
- `job_type`: new job class type: "Carving" (for smoothing and detailing)
- `unit_action_data_attack` (`unit_move_attackst`): identify flags

Documentation

- Lua API: documented existing `enum:next_item(index)` function

7.12.5 DFHack 50.12-r2.1

Fixes

- *control-panel*: properly auto-enable newly added bugfixes
- *fix/noexert-exhaustion*: fix typo in control panel registry entry which prevented the fix from being run when enabled
- *gui/suspendmanager*: fix script startup errors
- *orders*: don't intercept keyboard input for setting skill or labor restrictions on workshop workers tab when the player is setting the building nickname

Misc Improvements

- *gui/unit-syndromes*: make syndromes searchable by their display names (e.g. "necromancer")

7.12.6 DFHack 50.12-r2

New Tools

- *agitation-rebalance*: alter mechanics of irriation-related attacks so they are less constant and are more responsive to recent player bahavior
- *devel/block-borders*: (reinstated) highlights boundaries of map blocks or embark tile blocks
- *fix/noexert-exhaustion*: fix "Tired" NOEXERT units. Enabling via *gui/control-panel* prevents NOEXERT units from getting stuck in a "Tired" state
- *fix/ownership*: fix instances of multiple citizens claiming the same items, resulting in "Store owned item" job loops
- *fix/stuck-worship*: fix prayer so units don't get stuck in uninterruptible "Worship!" states
- *instruments*: provides information on how to craft the instruments used by the player civilization
- *modtools/if-entity*: (reinstated) modder's resource for triggering scripted content depending on the race of the loaded fort
- *modtools/item-trigger*: (reinstated) modder's resource for triggering scripted content when specific items are used

New Features

- *exterminate*: new "disintegrate" kill method that additionally destroys carried items
- *gui/settings-manager*: add import, export, and autoload for work details
- *logistics*: `autoretrain` will automatically assign trainers to your partially-trained (but not yet domesticated) livestock. this prevents children of partially-trained parents from reverting to wild if you don't notice they were born
- *orders*: add overlay for configuring labor and skill level restrictions for workshops
- *quickfort*: allow setting of workshop profile properties (e.g. labor, skill restrictions) from build blueprints
- *sort*: updated and reinstated military status/squad membership/burrow membership filter for work animal assignment screen
- *stocks*: add button/hotkey for removing empty categories from the stocks list

Fixes

- *autochop*: fix underestimation of log yield for cavern mushrooms
- *autoclothing*: don't produce clothes for dead units
- *caravan*: fix trade price calculations when the same item was requested for both import and export
- *catsplosion*: only cause pregnancies in adults
- *control-panel*: fix filtering not filtering when running the `list` command
- ***gui/launcher*:**
  - fix detection on Shift-Enter for running commands and autoclosing the launcher
  - fix history scanning (Up/Down arrow keys) being slow to respond when in minimal mode
- ***gui/notify*:**
  - prevent notification overlay from showing up in arena mode
  - don't zoom to forbidden depots for merchants ready to trade notification
- ***logistics*:**
  - don't melt/trade/dump empty containers that happen to be sitting on the stockpile unless the stockpile accepts those item types
  - don't send autotrade items to forbidden depots

Misc Improvements

- **Dreamfort:**
  - the four Craftsdwarf's workshops on the industry level are now specialized for Stonecrafting, Woodcrafting, Bone Carving, and miscellaneous tasks, respectively
  - update embark profile recommendations and example embark profile
- Many tools that previously only worked for citizens or only for dwarves now work for all citizens and residents, e.g. *fastdwarf*, *rejuvenate*, etc.
- When launched from the Steam client on Linux, both Dwarf Fortress and DFHack will be shown as "Running". This ensures that DF has proper accounting for Linux player usage.

- ***allneeds*:**
  - select a dwarf in the UI to see a summary of needs for just that dwarf
  - provide options for sorting the cumulative needs by different criteria
- *autobutcher*: prefer butchering partially trained animals and save fully domesticated animals to assist in wildlife domestication programs
- *autodump*: can now teleport items loosely stored in buildings (clutter)
- ***buildingplan*:**
  - remember player preference for whether unavailable materials should be hidden in the filter selection dialog
  - sort by available quantity by default int he filter selection dialog
- *cleaners*: protect farm plots when cleaning mud
- *control-panel*: enable tweaks quietly on fort load so we don't spam the console
- *devel/tile-browser*: simplify interface now that SDL automatically normalizes texture scale
- ***dwarfvet*:**
  - automatically unassign animals from pastures when they need treatment so they can make their way to the hospital. reassign them to their original pasture when treatment is complete.
  - ignore animals assigned to cages or restraints
- *exterminate*: make race name matching case and space insensitive
- *gui/gm-editor*: support opening engraved art for inspection
- ***gui/launcher*:**
  - add interface for browsing and filtering commands by tags
  - add support for history search (Alt-s hotkey) when in minimal mode
  - add support for the `clear` command and clearing the scrollback buffer
- *gui/notify*: Shift click or Shift Enter on a zoomable notification to zoom to previous target
- *gui/teleport*: add global Ctrl-Shift-T keybinding (only avaiable when DFHack mortal mode is disabled)
- *prioritize*: print out custom reaction and hauling jobs in the same format that is used for `prioritize` command arguments so the player can just copy and paste
- *suspendmanager*: improve performance when there are many active jobs
- *tweak*: add `quiet` option for silent enablement and disablement of tweaks

API

- `Units::getCitizens`: now includes residents by default
- `Units::isForgottenBeast`: property check for forgotten beasts
- `Units::isGreatDanger`: now includes forgotten beasts
- `Units::isResident`: property check for residents (as opposed to citizens)

Lua

- *helpdb*: `search_entries` now returns a match if *all* filters in the `include` list are matched. previous behavior was to match if *any* `include` filter matched.
- `dfhack.units.getCitizens`: now includes residents by default
- `dfhack.units.isForgottenBeast`: make new units method available to Lua
- `matinfo.decode`: now directly handles plant objects
- `widgets.Label`: `*pen` attributes can now either be a pen or a function that dynamically returns a pen

Structures

- `activity_event`: identify fields and type values
- `plant_tree_info`: define tree body and branch flags
- `plotinfo.hauling`: name fields related to the hauling route panel
- `unit`: identify and define many previously unknown fields, types, and enums

Documentation

- *Introduction and overview*: refresh getting started content
- *DFHack overlay dev guide*: updated examples and troubleshooting steps
- *Quickstart guide*: refresh quickstart guide

7.12.7 DFHack 50.12-r1.1

Fixes

- *sort*: fix crash when assigning work animals to units

Removed

- offline HTML rendered docs are no longer distributed with DFHack since they are randomly triggering Windows Defender antivirus heuristics. If you want to download DFHack docs for offline browsing, you can still get them from the Downloads link at https://dfhack.org/docs

7.12.8 DFHack 50.12-r1

Fixes

- *gui/design*: no longer comes up when Ctrl-D is pressed but other DFHack windows have focus
- *gui/notify*: persist notification settings when toggled in the UI

Misc Improvements

- *gui/launcher*: developer mode hotkey restored to Ctrl-D
- *sort*: squad assignment overlay rewritten for compatiblity with new vanilla data structures and screen layouts

Removed

- *burrow*: removed overlay 3D box select since it is now provided by the vanilla UI
- *sort*: removed Search widgets for screens that now have vanilla search

API

- `Gui::getWidget`: retrieve a vanilla DF widget by name or index

Lua

- `dfhack.gui.getWidgetChildren`: retrieve a list of child widgets for a given widget container
- `dfhack.gui.getWidget`: retrieve a vanilla DF widget by hierarchy path, with each step specified by a widget name or index

7.12.9 DFHack 50.11-r7

New Tools

- *add-thought*: (reinstated) add custom thoughts to a dwarf
- *combat-harden*: (reinstated) set a dwarf's resistence to being affected by visible corpses
- *devel/input-monitor*: interactive UI for debugging input issues
- *gui/notify*: display important notifications that vanilla doesn't support yet and provide quick zoom links to notification targets.
- *gui/petitions*: (reinstated) show outstanding (or all historical) petition agreements for guildhalls and temples
- *list-waves*: (reinstated) show migration wave information
- *make-legendary*: (reinstated) make a dwarf legendary in specified skills
- *pet-uncapper*: (reinstated, renamed from `petcapRemover`) allow pets to breed beyond the default population cap of 50
- *tweak*: (reinstated) a collection of small bugfixes and gameplay tweaks
- *undump-buildings*: (reinstated) remove dump designation from in-use building materials

New Features

- *cleanowned*: Add a "nodump" option to allow for confiscating items without dumping
- *tweak*: Add "flask-contents", makes flasks/vials/waterskins be named according to their contents

Fixes

- *autoclothing*: Fix enabled behavior
- *caravan*: display book and scroll titles in the goods and trade dialogs instead of generic scroll descriptions
- *dig-now*: fix digging stairs in the surface sometimes creating underworld gates.
- *dig*: overlay that shows damp designations in ASCII mode now propertly highlights tiles that are damp because of an aquifer in the layer above
- *fix/retrieve-units*: prevent pulling in duplicate units from offscreen
- *gui/blueprint*: changed hotkey for setting blueprint origin tile so it doesn't conflict with default map movement keys
- *gui/control-panel*: fix error when toggling autostart settings
- *gui/design*: clicking the center point when there is a design mark behind it will no longer simultaneously enter both mark dragging and center dragging modes. Now you can click once to move the shape, and click twice to move only the mark behind the center point.
- *item*: avoid error when scanning items that have no quality rating (like bars and other construction materials)
- *source*: fix issue where removing sources would make some other sources inactive
- *strangemood*: correctly recognize Stonecutter and Stone Carver as moodable skills, move the Mason's boosted mood chance to the Stone Carver, and select Fell/Macabre based on long-term stress
- ***warn-stranded*:**
  - don't complain about units that aren't on the map (e.g. soldiers out on raids)
  - when there was at least one truly stuck unit and miners were actively mining, the miners were also confusingly shown in the stuck units list
- `gui.View:getMouseFramePos`: function now detects the correct coordinates even when the widget is nested within other frames
- `Gui::makeAnnouncement`, `Gui::autoDFAnnouncement`: don't display popup for all announcement types
- `Gui::revealInDwarfmodeMap`: properly center the zoom even when the target tile is near the edge of the map
- `Units::getVisibleName`: don't reveal the true identities of units that are impersonating other historical figures

Misc Improvements

- *autonestbox*: assign egg layers to the nestbox they have chosen if they have already chosen a nestbox
- *buildingplan*: use closest matching item rather than newest matching item
- *caravan*: move goods to trade depot dialog now allocates more space for the display of the value of very expensive items
- *exportlegends*: make progress increase smoothly over the entire export and increase precision of progress percentage

- *extinguish*: allow selecting units/items/buildings in the UI to target them for extinguishing; keyboard cursor is only required for extinguishing map tiles that cannot be selected any other way
- *gui/autobutcher*: ask for confirmation before zeroing out targets for all races
- *gui/mod-manager*: will automatically unmark the default mod profile from being the default if it fails to load (due to missing or incompatible mods)
- ***gui/quickfort***:
  - can now dynamically adjust the dig priority of tiles designated by dig blueprints
  - can now opt to apply dig blueprints in marker mode
- ***item***:
  - change syntax so descriptions can be searched for without indicating the `--description` option. e.g. it's now `item count royal` instead of `item count --description royal`
  - add `--verbose` option to print each item as it is matched
- *probe*: act on the selected building/unit instead of requiring placement of the keyboard cursor for `bprobe` and `cprobe`
- *regrass*: also regrow depleted cavern moss
- ***zone***:
  - animal assignment dialog now shows distance to pasture/cage and allows sorting by distance
  - animal assignment dialog shows number of creatures assigned to this pasture/cage/etc.

Removed

- *gui/create-tree*: replaced by *gui/sandbox*
- *gui/manager-quantity*: the vanilla UI can now modify manager order quantities after creation
- *warn-starving*: combined into *gui/notify*
- *warn-stealers*: combined into *gui/notify*

API

- Gui focus strings will now include `dwarfmode/Default` if the only other panel open is the Squads panel
- Gui module Announcement functions now use DF's new announcement alert system
- `Gui::addCombatReport`, `Gui::addCombatReportAuto`: add versions that take `report *` instead of report vector index
- `Gui::MTB_clean`, `Gui::MTB_parse`, `Gui::MTB_set_width`: new functions for manipulating `markup_text_boxst`
- `Gui::revealInDwarfmodeMap`: unfollow any currently followed units/items so the viewport doesn't just jump back to where it was
- `toupper_cp437(char)`, `tolower_cp437(char)`: new `MiscUtils` functions, return a char with case changed, respecting CP437
- `toUpper`, `toLower`: `MiscUtils` functions renamed to `toUpper_cp437` and `toLower_cp437`, CP437 compliant

Lua

- Overlay framework now respects `active` and `visible` widget attributes
- `dfhack.gui` announcement functions use default arguments when omitted
- `dfhack.units.getCitizens` now only returns units that are on the map
- `dfhack.upperCp437(string)`, `dfhack.lowerCp437(string)`: new functions, return string with all chars changed, respecting CP437 code page

Structures

- `buildings_other`: add correct types for civzone building vectors
- `job_skill`: correct `moodable` property for several professions

7.12.10 DFHack 50.11-r6

New Features

- *zone*: Add overlay for toggling butchering/gelding/adoption/taming options in animal "Overview" tabs

Fixes

- ***dig-now*:**
  - remove diagonal ramps rendered unusable by digging
  - fix error propagating "light" and "outside" properties to newly exposed tiles when piercing the surface
- *item*: fix missing item categories when using `--by-type`
- *makeown*: fix error when adopting units that need a historical figure to be created
- *sort*: fix potential crash when switching between certain info tabs
- *suspendmanager*: overlays for suspended building info panels no longer disappear when another window has focus

Misc Improvements

- *autonestbox*: don't automatically assign partially trained egg-layers to nestboxes if they don't have an ongoing trainer assigned since they might revert to wild
- *buildingplan*: replace `[edit filters]` button in planner overlay with abbreviated filter information
- *reveal*: automatically reset saved map state when a new save is loaded

Removed

- `nopause`: functionality has moved to *spectate*

API

- `Gui::getAnyJob`: get the job associated with the selected game element (item, unit, workshop, etc.)
- `Gui::getAnyWorkshopJob`: get the first job associated with the selected workshop
- `Units::assignTrainer`: assign a trainer to a trainable animal
- `Units::unassignTrainer`: unassign a trainer from an animal

Lua

- `dfhack.gui.getAnyJob`: expose API to Lua
- `dfhack.gui.getAnyWorkshopJob`: expose API to Lua
- `dfhack.units.assignTrainer`: expose API to Lua
- `dfhack.units.isTamable`: return false for invaders to match vanilla logic
- `dfhack.units.unassignTrainer`: expose API to Lua

Structures

- `soundst`: fix alignment

7.12.11 DFHack 50.11-r5

New Tools

- *control-panel*: new commandline interface for control panel functions
- *gui/biomes*: visualize and inspect biome regions on the map
- ***gui/embark-anywhere*:**
  - new keybinding (active when choosing an embark site): Ctrl-A
  - bypass those pesky warnings and embark anywhere you want to
- *gui/reveal*: temporarily unhide terrain and then automatically hide it again when you're ready to unpause
- *gui/teleport*: mouse-driven interface for selecting and teleporting units
- *item*: perform bulk operations on groups of items.
- *uniform-unstick*: (reinstated) force squad members to drop items that they picked up in the wrong order so they can get everything equipped properly

New Features

- *gui/mass-remove*: new global keybinding: Ctrl-M while on the fort map
- *gui/settings-manager*: save and load embark difficulty settings and standing orders; options for auto-load on new embark
- *sort*: search and sort for the "choose unit to elevate to the barony" screen. units are sorted by the number of item preferences they have and the units are annotated with the items that they have preferences for
- *uniform-unstick*: add overlay to the squad equipment screen to show a equipment conflict report and give you a one-click button to (attempt to) fix
- *zone*: add button to location details page for retiring unused locations

Fixes

- DFHack tabs (e.g. in *gui/control-panel*) are now rendered correctly when there are certain vanilla screen elements behind them
- Dreamfort: fix holes in the "Inside+" burrow on the farming level (burrow autoexpand is interrupted by the pre-dug miasma vents to the surface)
- When passing map movement keys through to the map from DFHack tool windows, also pass fast z movements (shift-scroll by default)
- *ban-cooking*: fix banning creature alcohols resulting in error
- ***buildingplan*:**
  - when you save a game and load it again, newly planned buildings are now correctly placed in line after existing planned buildings of the same type
  - treat items in wheelbarrows as unavailable, just as vanilla DF does. Make sure the *fix/empty-wheelbarrows* fix is enabled so those items aren't permanently unavailable!
  - show correct number of materials required when laying down areas of constructions and some of those constructions are on invalid tiles
- *caravan*: ensure items are marked for trade when the move trade goods dialog is closed even when they were selected and then the list filters were changed such that the items were no longer actively shown
- *confirm*: properly detect clicks on the remove zone button even when the unit selection screen is also open (e.g. the vanilla assign animal to pasture panel)
- *empty-bin*: now correctly sends ammunition in carried quivers to the tile underneath the unit instead of teleporting them to an invalid (or possibly just far away) location
- ***fastdwarf*:**
  - prevent units from teleporting to inaccessible areas when in teledwarf mode
  - allow units to meander and satisfy needs when they have no current job and teledwarf mode is enabled
- *getplants*: fix crash when processing mod-added plants with invalid materials
- ***gui/design*:**
  - fix incorrect highlight when box selecting area in ASCII mode
  - fix incorrect dimensions being shown when you're placing a stockpile, but a start coordinate hasn't been selected yet
- *misery*: fix error when changing the misery factor

- *quickfort*: if a blueprint specifies an up/down stair, but the tile the blueprint is applied to cannot make an up stair (e.g. it has already been dug out), still designate a down stair if possible
- *reveal*: now avoids revealing blocks that contain divine treasures, encased horrors, and deep vein hollows (so the surprise triggers are not triggered prematurely)
- ***sort*:**
  - fix mouse clicks falling through the squad assignment overlay panel when clicking on the panel but not on a clickable widget
  - fix potential crash when removing jobs directly from the Tasks info screen
- *source*: water and magma sources and sinks now persist with fort across saves and loads
- *stonesense*: fix crash in cleanup code after mega screenshot (Ctrl-F5) completes; however, the mega screenshot will still make stonesense unresponsive. close and open the stonesense window to continue using it.
- *suspendmanager*: correctly handle building collisions with smoothing designations when the building is on the edge of the map
- *warn-stranded*: don't warn for citizens who are only transiently stranded, like those on stepladders gathering plants or digging themselves out of a hole
- `Maps::getBiomeType`, `Maps::getBiomeTypeWithRef`: fix identification of tropical oceans

Misc Improvements

- Dreamfort: put more chairs adjacent to each other to make the tavern more "social"
- The "PAUSE FORCED" badge will blink briefly to draw attention if the player attempts to unpause when a DFHack tool window requires the game to be paused
- wherever units are listed in DFHack tools, properties like "agitated" or (-trained-) are now shown
- *autochop*: better error output when target burrows are not specified on the commandline
- *autoclothing* : now does not consider worn (x) clothing as usable/available; reduces overproduction when using *tailor* at same time
- *buildingplan*: add option for preventing constructions from being planned on top of existing constructions (e.g. don't build floors on top of floors)
- *burrow*: flood fill now requires an explicit toggle before it is enabled to help prevent accidental flood fills
- ***confirm*:**
  - updated confirmation dialogs to use clickable widgets and draggable windows
  - added confirmation prompt for right clicking out of the trade agreement screen (so your trade agreement selections aren't lost)
  - added confirmation prompts for irreversible actions on the trade screen
  - added confirmation prompt for deleting a uniform
  - added confirmation prompt for convicting a criminal
  - added confirmation prompt for re-running the embark site finder
  - added confirmation prompt for reassigning or clearing zoom hotkeys
  - added confirmation prompt for exiting the uniform customization page without saving
- *fastdwarf*: now saves its state with the fort

- *fix/stuck-instruments*: now handles instruments that are left in the "in job" state but that don't have any actual jobs associated with them
- *gui/autobutcher*: interface redesigned to better support mouse control
- ***gui/control-panel*:**
  - reduce frequency for *warn-stranded* check to once every 2 days
  - tools are now organized by type: automation, bugfix, and gameplay
- ***gui/launcher*:**
  - now persists the most recent 32KB of command output even if you close it and bring it back up
  - make autocomplete case insensitive
- ***gui/mass-remove*:**
  - can now differentiate planned constructions, stockpiles, and regular buildings
  - can now remove zones
  - can now cancel removal for buildings and constructions
- *gui/quickcmd*: clickable buttons for command add/remove/edit operations
- *orders*: reduce prepared meal target and raise booze target in `basic` importable orders in the orders library
- ***sort*:**
  - add "Toggle all filters" hotkey button to the squad assignment panel
  - rename "Weak mental fortitude" filter to "Dislikes combat", which should be more understandable
- *uniform-unstick*: warn if a unit belongs to a squad from a different site (can happen with migrants from previous forts)
- *warn-stranded*: center the screen on the unit when you select one in the list
- *work-now*: now saves its enabled status with the fort
- ***zone*:**
  - add include/only/exclude filter for juveniles to the pasture/pit/cage/restraint assignment screen
  - show geld status and custom profession (if set, it's the lower editable line in creature description) in pasture/pit/cage/restraint assignment screen

Removed

- *channel-safely*: (temporarily) removed due to stability issues with the underlying DF API
- `persist-table`: replaced by new `dfhack.persistent` API

API

- New plugin API for saving and loading persistent data. See plugins/examples/skeleton.cpp and plugins/examples/persistent_per_save_example.cpp for details
- Plugin ABI (binary interface) version bump! Any external plugins must be recompiled against this version of DFHack source code in order to load.
- `capitalize_string_words`: new `MiscUtils` function, returns string with all words capitalized
- `Constructions::designateRemove`: no longer designates the non-removable "pseudo" construtions that represent the top of walls
- `grab_token_string_pos`: new `MiscUtils` function, used for parsing tokens
- `Items`: add item melting logic `canMelt(item)`, `markForMelting(item)`, and `cancelMelting(item)`
- **`Persistence:`**
  - persistent keys are now namespaced by an entity_id (e.g. a player fort site ID)
  - data is now stored one file per entity ID (plus one for the global world) in the DF savegame directory
- `random_index`, `vector_get_random`: new `MiscUtils` functions, for getting a random entry in a vector
- `Units.isDanger`: now returns true for agitated wildlife
- **`World:`**
  - `GetCurrentSiteId()` returns the loaded fort site ID (or -1 if no site is loaded)
  - `IsSiteLoaded()` check to detect if a site (e.g. a player fort) is active (as opposed to the world or a map)
  - `AddPersistentData` and related functions replaced with `AddPersistentSiteData` and `AddPersistentWorldData` equivalents

Lua

- `dfhack.capitalizeStringWords`: new function, returns string with all words capitalized
- `dfhack.isSiteLoaded`: returns whether a site (e.g. a player fort) is loaded
- `dfhack.items`: access to `canMelt(item)`, `markForMelting(item)`, and `cancelMelting(item)` from `Items` module
- `dfhack.persistent`: new, table-driven API for easier world- and site-associated persistent storage. See the Lua API docs for details.
- `dfhack.world.getCurrentSite`: returns the `df.world_site` instance of the currently loaded fort
- `widgets.Divider`: linear divider to split an existing frame; configurable T-junction edges and frame style matching

Structures

- `alert_button_announcement_id`: now int32_t vector (contains report ids)
- `announcement_alertst`: defined
- `announcement_alert_type`: enum defined
- `announcement_type`: added `alert_type` enum attribute
- `feature_init_flags`: more enum values defined
- `markup_text_boxst`: updated based on information from Bay12
- `markup_text_linkst`, `markup_text_wordst`, `script_environmentst`: defined
- `occupation`: realigned
- `plotinfost`: `unk23c8_flags` renamed to `flags`, updated based on information from Bay12
- `service_orderst`: type defined
- `service_order_type`: enum defined
- `soundst`: defined
- `viewscreen_choose_start_sitest`: fix structure of warning flags – convert series of bools to a proper bitmask
- `world_raws`: `unk_v50_1`, `unk_v50_2`, `unk_v50_3` renamed to `text_set`, `music`, `sound`

Documentation

- DFHack developer's guide updated, with refreshed *Architecture diagrams*
- UTF-8 text in tool docs is now properly displayed in-game in *gui/launcher* (assuming that it can be converted to cp-437)
- *Installing*: Add installation instructions for wineskin on Mac
- *DFHack modding guide*: Add examples for script-only and blueprint-only mods that you can upload to DF's Steam Workshop

7.12.12 DFHack 50.11-r4

New Tools

- *build-now*: (reinstated) instantly complete unsuspended buildings that are ready to be built

Fixes

- RemoteServer: don't shut down the socket prematurely, allowing continuing connections from, for example, dfhack-run
- *buildingplan*: fix choosing the wrong mechanism (or something that isn't a mechanism) when linking a lever and manually choosing a mechanism, but then canceling the selection
- *combine*: prevent stack sizes from growing beyond quantities that you would normally see in vanilla gameplay
- *gui/design*: Center dragging shapes now track the mouse correctly

- ***sort*:**
  - fix potential crash when exiting and re-entering a creatures subtab with a search active
  - prevent keyboard keys from affecting the UI when search is active and multiple keys are hit at once
- *tailor*: fix corner case where existing stock was being ignored, leading to over-ordering

Misc Improvements

- ***buildingplan*:**
  - save magma safe mechanisms for when magma safety is requested when linking levers and pressure plates to targets
  - when choosing mechanisms for linking levers/pressure plates, filter out unreachable mechanisms
- *caravan*: enable searching within containers in trade screen when in "trade bin with contents" mode
- *sort*: when searching on the Tasks tab, also search the names of the things the task is associated with, such as the name of the stockpile that an item will be stored in

7.12.13 DFHack 50.11-r3

New Tools

- *burrow*: (reinstated) automatically expand burrows as you dig
- *sync-windmills*: synchronize or randomize movement of active windmills
- *trackstop*: (reimplemented) integrated overlay for changing track stop and roller settings after construction

New Features

- *buildingplan*: allow specific mechanisms to be selected when linking levers or pressure plates
- *burrow*: integrated 3d box fill and 2d/3d flood fill extensions for burrow painting mode
- *fix/dead-units*: gained ability to scrub dead units from burrow membership lists
- *gui/design*: show selected dimensions next to the mouse cursor when designating with vanilla tools, for example when painting a burrow or designating digging
- *prospector*: can now give you an estimate of resources from the embark screen. hover the mouse over a potential embark area and run *prospector*.
- *quickfort*: new `burrow` blueprint mode for designating or manipulating burrows
- *sort*: military and burrow membership filters for the burrow assignment screen
- *unforbid*: now ignores worn and tattered items by default (X/XX), use -X to bypass

Fixes

- RemoteServer: continue to accept connections as long as the listening socket is valid instead of closing the socket after the first disconnect
- *buildingplan*: overlay and filter editor gui now uses ctrl-d to delete the filter to avoid conflict with increasing the filter's minimum quality (shift-x)
- *caravan*: price of vermin swarms correctly adjusted down. a stack of 10000 bees is worth 10, not 10000
- *emigration*: fix clearing of work details assigned to units that leave the fort
- *gui/unit-syndromes*: show the syndrome names properly in the UI
- *sort*: when filtering out already-established temples in the location assignment screen, also filter out the "No specific deity" option if a non-denominational temple has already been established
- *stockpiles*: hide configure and help buttons when the overlay panel is minimized
- *tailor*: fix crash on Linux where scanned unit is wearing damaged non-clothing (e.g. a crown)

Misc Improvements

- ***buildingplan***:
  - display how many items are available on the planner panel
  - make it easier to build single-tile staircases of any shape (up, down, or up/down)
- *Dreamfort*: Inside+ and Clearcutting burrows now automatically created and managed
- ***sort***:
  - allow searching by profession on the squad assignment page
  - add search for places screens
  - add search for work animal assignment screen; allow filtering by miltary/squad/civilian/burrow
  - on the squad assignment screen, make effectiveness and potential ratings use the same scale so effectiveness is always less than or equal to potential for a given unit. this way you can also tell when units are approaching their maximum potential
  - new overlay on the animal assignment screen that shows how many work animals each visible unit already has assigned to them
- *warn-stranded*: don't warn for units that are temporarily on unwalkable tiles (e.g. as they pass under a waterfall)

Removed

- ***gui/control-panel***:
  - removed always-on system services from the `System` tab: *buildingplan*, *confirm*, *logistics*, and *overlay*. The base services should not be turned off by the player. Individual confirmation prompts can be managed via *gui/confirm*, and overlays (including those for *buildingplan* and *logistics*) are managed on the control panel `Overlays` tab.
  - removed *autolabor* from the `Fort` and `Autostart` tabs. The tool does not function correctly with the new labor types, and is causing confusion. You can still enable *autolabor* from the commandline with `enable autolabor` if you understand and accept its limitations.

API

- `Buildings::completebuild`: used to link a newly created building into the world
- `Burrows::setAssignedUnit`: now properly handles inactive burrows
- `Gui::getMousePos`: now takes an optional `allow_out_of_bounds` parameter so coordinates can be returned for mouse positions outside of the game map (i.e. in the blank space around the map)
- `Gui::revealInDwarfmodeMap`: gained `highlight` parameter to control setting the tile highlight on the zoom target
- `Maps::getWalkableGroup`: get the walkability group of a tile
- `Units::getReadableName`: now returns the *untranslated* name

Lua

- `dfhack.buildings.completebuild`: expose new module API
- `dfhack.gui.getMousePos`: support new optional `allow_out_of_bounds` parameter
- `dfhack.gui.revealInDwarfmodeMap`: gained `highlight` parameter to control setting the tile highlight on the zoom target
- `dfhack.maps.getWalkableGroup`: get the walkability group of a tile
- `gui.FRAME_THIN`: a panel frame suitable for floating tooltips

Structures

- `burrow`: add new graphics mode texture and color fields
- `job_item_flags3`: identify additional flags

Documentation

- Document the Lua API for the `dfhack.world` module

7.12.14 DFHack 50.11-r2

New Tools

- *add-recipe*: (reinstated) add reactions to your civ (e.g. for high boots if your civ didn't start with the ability to make high boots)
- *burial*: (reinstated) create tomb zones for unzoned coffins
- *fix/corrupt-jobs*: prevents crashes by automatically removing corrupted jobs
- *preserve-tombs*: keep tombs assigned to units when they die
- *spectate*: (reinstated) automatically follow dwarves, cycling among interesting ones

New Scripts

- *warn-stranded*: new repeatable maintenance script to check for stranded units, similar to *warn-starving*

New Features

- *burial*: new options to configure automatic burial and limit scope to the current z-level
- ***drain-aquifer***:
  - gained ability to drain just above or below a certain z-level
  - new option to drain all layers except for the first N aquifer layers, in case you want some aquifer layers but not too many
- *gui/control-panel*: `drain-aquifer --top 2` added as an autostart option
- *logistics*: `automelt` now optionally supports melting masterworks; click on gear icon on *stockpiles* overlay frame
- ***sort***:
  - new search widgets for Info panel tabs, including all "Creatures" subtabs, all "Objects" subtabs, "Tasks", candidate assignment on the "Noble" subtab, and the "Work details" subtab under "Labor"
  - new search and filter widgets for the "Interrogate" and "Convict" screens under "Justice"
  - new search widgets for location selection screen (when you're choosing what kind of guildhall or temple to dedicate)
  - new search widgets for burrow assignment screen and other unit assignment dialogs
  - new search widgets for artifacts on the world/raid screen
  - new search widgets for slab engraving menu; can filter for only units that need a slab to prevent rising as a ghost
- *stocks*: hotkey for collapsing all categories on stocks screen

Fixes

- ***buildingplan***:
  - remove bars of ash, coal, and soap as valid building materials to match v50 rules
  - fix incorrect required items being displayed sometimes when switching the planner overlay on and off
- *dwarfvet*: fix invalid job id assigned to `Rest` job, which could cause crashes on reload
- *full-heal*: fix removal of corpse after resurrection
- *gui/sandbox*: fix scrollbar moving double distance on click
- *hide-tutorials*: fix the embark tutorial prompt sometimes not being skipped
- *sort*: don't count mercenaries as appointed officials in the squad assignment screen
- *suspendmanager*: fix errors when constructing near the map edge
- *toggle-kbd-cursor*: clear the cursor position when disabling, preventing the game from sometimes jumping the viewport around when cursor keys are hit
- ***zone***:

  - races without specific child or baby names will now get generic child/baby names instead of an empty string
  - don't show animal assignment link for cages and restraints linked to dungeon zones (which aren't normally assignable)

Misc Improvements

- Help icons added to several complex overlays. clicking the icon runs *gui/launcher* with the help text in the help area
- ***buildingplan*:**
  - support filtering cages by whether they are occupied
  - show how many items you need to make when planning buildings
- *gui/gm-editor*: for fields with primitive types, change from click to edit to click to select, double-click to edit. this should help prevent accidental modifications to the data and make hotkeys easier to use (since you have to click on a data item to use a hotkey on it)
- *gui/overlay*: filter overlays by current context so there are fewer on the screen at once and you can more easily click on the one you want to reposition
- *orders*: `recheck` command now only resets orders that have conditions that can be rechecked
- *overlay*: allow `overlay_onupdate_max_freq_seconds` to be dynamically set to 0 for a burst of high-frequency updates
- *prioritize*: refuse to automatically prioritize dig and smooth/carve job types since it can break the DF job scheduler; instead, print a suggestion that the player use specialized units and vanilla designation priorities
- *quickfort*: now allows constructions to be built on top of constructed floors and ramps, just like vanilla. however, to allow blueprints to be safely reapplied to the same area, for example to fill in buildings whose constructions were canceled due to lost items, floors will not be rebuilt on top of floors and ramps will not be rebuilt on top of ramps
- *sort*: added help button for squad assignment search/filter/sort
- *tailor*: now adds to existing orders if possilbe instead of creating new ones
- *zone*: animals trained for war or hunting are now labeled as such in animal assignment screens

Removed

- `FILTER_FULL_TEXT`: moved from `gui.widgets` to `utils`; if your full text search preference is lost, please reset it in *gui/control-panel*

API

- added `Items::getCapacity`, returns the capacity of an item as a container (reverse-engineered), needed for *combine*

Lua

- added `dfhack.items.getCapacity` to expose the new module API
- added `GRAY` color aliases for `GREY` colors
- `utils.search_text`: text search routine (generalized from internal `widgets.FilteredList` logic)

Structures

- add new globals: `translate_name`, `buildingst_completebuild`
- `artifact_rumor_locationst`: defined
- `viewscreen_worldst`: defined types for `view_mode` and `artifacts_arl` fields
- `world_view_mode_type`: defined

Documentation

- unavailable tools are no longer listed in the tag indices in the online docs

7.12.15 DFHack 50.11-r1

New Tools

- *startdwarf*: (reinstated) set number of starting dwarves
- *tubefill*: (reinstated) replenishes mined-out adamantine

New Features

- A new searchable, sortable, filterable dialog for selecting items for display on pedestals and display cases
- *startdwarf*: overlay scrollbar so you can scroll through your starting dwarves if they don't all fit on the screen

Fixes

- EventManager: Unit death event no longer misfires on units leaving the map
- *autolabor*: ensure vanilla work details are reinstated when the fort or the plugin is unloaded
- *suspendmanager*: fixed a bug where floor grates, bars, bridges etc. wouldn't be recognised as walkable, leading to unnecessary suspensions in certain cases.
- `dfhack.TranslateName()`: fixed crash on certain invalid names, which affected *warn-starving*

Misc Improvements

- **EventManager:**
  - guard against potential iterator invalidation if one of the event listeners were to modify the global data structure being iterated over
  - for `onBuildingCreatedDestroyed` events, changed firing order of events so destroyed events come before created events
- *devel/inspect-screen*: display total grid size for UI and map layers
- ***dig*:**
  - designate only visible tiles by default, and use "auto" dig mode for following veins
  - added options for designating only current z-level, this z-level and above, and this z-level and below
- ***hotkeys*:**
  - make the DFHack logo brighten on hover in ascii mode to indicate that it is clickable
  - use vertical bars instead of "!" symbols for the DFHack logo in ascii mode to make it easier to read
- *suspendmanager*: now suspends constructions that would cave-in immediately on completion

Lua

- mouse key events are now aligned with internal DF semantics: `_MOUSE_L` indicates that the left mouse button has just been pressed and `_MOUSE_L_DOWN` indicates that the left mouse button is being held down. similarly for `_MOUSE_R` and `_MOUSE_M`. 3rd party scripts may have to adjust.

Structures

- add new global: `start_dwarf_count`

7.12.16 DFHack 50.10-r1

Fixes

- 'fix/general-strike: fix issue where too many seeds were getting planted in farm plots
- Linux launcher: allow Steam Overlay and game streaming to function
- *autobutcher*: don't ignore semi-wild units when marking units for slaughter

Misc Improvements

- 'sort': Improve combat skill scale thresholds

7.12.17 DFHack 50.09-r4

New Features

- *dig*: new overlay for ASCII mode that visualizes designations for smoothing, engraving, carving tracks, and carving fortifications

Fixes

- *buildingplan*: make the construction dimensions readout visible again
- *gui/mod-manager*: don't continue to display overlay after the raws loading progress bar appears
- *seedwatch*: fix a crash when reading data saved by very very old versions of the plugin

Misc Improvements

- *autofish*: changed `--raw` argument format to allow explicit setting to on or off
- *caravan*: move goods to depot screen can now see/search/trade items inside of barrels and pots
- *gui/launcher*: show tagged tools in the autocomplete list when a tag name is typed
- ***sort*:**
  - add sort option for training need on squad assignment screen
  - filter mothers with infants, units with weak mental fortitude, and critically injured units on the squad assignment screen
  - display a rating relative to the current sort order next to the visible units on the squad assignment screen

API

- *overlay*: overlay widgets can now declare a `version` attribute. changing the version of a widget will reset its settings to defaults. this is useful when changing the overlay layout and old saved positions will no longer be valid.

Lua

- `argparse.boolean`: convert arguments to lua boolean values.

Structures

- Identified a number of previously anonymous virtual methods in `itemst`

Documentation

- add instructions for downloading development builds to the `Installing` page

7.12.18 DFHack 50.09-r3

New Tools

- *devel/scan-vtables*: scan and dump likely vtable addresses (for memory research)
- *hide-interface*: hide the vanilla UI elements for clean screenshots or laid-back fortress observing
- *hide-tutorials*: hide the DF tutorial popups; enable in the System tab of *gui/control-panel*
- *set-orientation*: tinker with romantic inclinations (reinstated from back catalog of tools)

New Features

- *buildingplan*: one-click magma/fire safety filter for planned buildings
- *exportlegends*: new overlay that integrates with the vanilla "Export XML" button. Now you can generate both the vanilla export and the extended data export with a single click!
- *sort*: search, sort, and filter for squad assignment screen
- *zone*: advanced unit assignment screens for cages, restraints, and pits/ponds

Fixes

- **Core:**
  - reload scripts in mods when a world is unloaded and immediately loaded again
  - fix text getting added to DFHack text entry widgets when Alt- or Ctrl- keys are hit
- *autobutcher*: fix `ticks` commandline option incorrectly rejecting positive integers as valid values
- *buildingplan*: ensure selected barrels and buckets are empty (or at least free of lye and milk) as per the requirements of the building
- ***caravan*:**
  - corrected prices for cages that have units inside of them
  - correct price adjustment values in trade agreement details screen
  - apply both import and export trade agreement price adjustments to items being both bought or sold to align with how vanilla DF calculates prices
  - cancel any active TradeAtDepot jobs if all caravans are instructed to leave
- ***emigration*:**
  - fix errors loading forts after dwarves assigned to work details or workshops have emigrated
  - fix citizens sometimes "emigrating" to the fortress site
- *fix/retrieve-units*: fix retrieved units sometimes becoming duplicated on the map
- *gui/launcher*, *gui/gm-editor*: recover gracefully when the saved frame position is now offscreen
- *gui/sandbox*: correctly load equipment materials in modded games that categorize non-wood plants as wood

- *orders*: prevent import/export overlay from appearing on the create workorder screen
- *quickfort*: cancel old dig jobs that point to a tile when a new designation is applied to the tile
- *seedwatch*: ignore unplantable tree seeds
- *starvingdead*: ensure sieges end properly when undead siegers starve
- ***suspendmanager*:**
  - Fix the overlay enabling/disabling *suspendmanager* unexpectedly
  - improve the detection on "T" and "+" shaped high walls
- *tailor*: remove crash caused by clothing items with an invalid `maker_race`
- `dialogs.MessageBox`: fix spacing around scrollable text

Misc Improvements

- Surround DFHack-specific UI elements with square brackets instead of red-yellow blocks for better readability
- *autobutcher*: don't mark animals for butchering if they are already marked for some kind of training (war, hunt)
- *caravan*: optionally display items within bins in bring goods to depot screen
- *createitem*: support creating items inside of bags
- *devel/lsmem*: added support for filtering by memory addresses and filenames
- *gui/design*: change "auto commit" hotkey from `c` to `Alt-c` to avoid conflict with the default keybinding for z-level down
- ***gui/gm-editor*:**
  - hold down shift and right click to exit, regardless of how many substructures deep you are
  - display in the title bar whether the editor window is scanning for live updates
- *gui/liquids*: support removing river sources by converting them into stone floors
- *gui/quickfort*: blueprint details screen can now be closed with Ctrl-D (the same hotkey used to open the details)
- *hotkeys*: don't display DFHack logo in legends mode since it covers up important interface elements. the Ctrl-Shift-C hotkey to bring up the menu and the mouseover hotspot still function, though.
- *quickfort*: linked stockpiles and workshops can now be specified by ID instead of only by name. this is mostly useful when dynamically generating blueprints and applying them via the *quickfort* API
- *sort*: animals are now sortable by race on the assignment screens
- *suspendmanager*: display a different color for jobs suspended by suspendmanager

API

- *RemoteFortressReader*: add a `force_reload` option to the GetBlockList RPC API to return blocks regardless of whether they have changed since the last request
- `Gui`: `getAnyStockpile` and `getAnyCivzone` (along with their `getSelected` variants) now work through layers of ZScreens. This means that they will still return valid results even if a DFHack tool window is in the foereground.
- `Items::getValue()`: remove `caravan_buying` parameter since the identity of the selling party doesn't actually affect the item value

- `Units`: new animal propery check functions `isMarkedForTraining(unit)`, `isMarkedForTaming(unit)`, `isMarkedForWarTraining(unit)`, and `isMarkedForHuntTraining(unit)`

Lua

- `dfhack.gui`: new `getAnyCivZone` and `getAnyStockpile` functions; also behavior of `getSelectedCivZone` and `getSelectedStockpile` functions has changes as per the related API notes
- `dfhack.items.getValue()`: remove `caravan_buying` param as per C++ API change
- `dfhack.screen.readTile()`: now populates extended tile property fields (like `top_of_text`) in the returned `Pen` object
- `dfhack.units`: new animal propery check functions `isMarkedForTraining(unit)`, `isMarkedForTaming(unit)`, `isMarkedForWarTraining(unit)`, and `isMarkedForHuntTraining(unit)`
- `new()`: improved error handling so that certain errors that were previously uncatchable (creating objects with members with unknown vtables) are now catchable with `pcall()`
- `widgets.BannerPanel`: panel with distinctive border for marking DFHack UI elements on otherwise vanilla screens
- `widgets.Panel`: new functions to override instead of setting corresponding properties (useful when subclassing instead of just setting attributes): `onDragBegin`, `onDragEnd`, `onResizeBegin`, `onResizeEnd`

Structures

- Added `global_table` global and corresponding `global_table_entry` type
- `help_context_type`: fix typo in enum name: `EMBARK_TUTORIAL_CHICE` -> `EMBARK_TUTORIAL_CHOICE`
- `plotinfo`: name the fields related to tutorial popups
- `viewscreen_legendsst`: realign structure
- `viewscreen_new_arenast`: added (first appeared in 50.06)

7.12.19 DFHack 50.09-r2

New Plugins

- *3dveins*: reinstated for v50, this plugin replaces vanilla DF's blobby vein generation with veins that flow smoothly and naturally between z-levels
- *dig*: new `dig.asciiwarmdamp` overlay that highlights warm and damp tiles when in ASCII mode. there is no effect in graphics mode since the tiles are already highlighted there
- *dwarfvet*: reinstated and updated for v50's new hospital mechanics; allow your animals to have their wounds treated at hospitals
- *zone*: new searchable, sortable, filterable screen for assigning units to pastures

New Scripts

- *caravan*: new trade screen UI replacements for bringing goods to trade depot and trading
- *fix/empty-wheelbarrows*: new script to empty stuck rocks from all wheelbarrows on the map

Fixes

- Fix extra keys appearing in DFHack text boxes when shift (or any other modifier) is released before the other key you were pressing
- *gui/autodump*: when "include items claimed by jobs" is on, actually cancel the job so the item can be teleported
- *gui/create-item*: when choosing a citizen to create the chosen items, avoid choosing a dead citizen
- *gui/gm-unit*: fix commandline processing when a unit id is specified
- ***logistics*:**
  - don't autotrain domestic animals brought by invaders (they'll get attacked by friendly creatures as soon as you let them out of their cage)
  - don't bring trade goods to depot if the only caravans present are tribute caravans
  - fix potential crash when removing stockpiles or turning off stockpile features
- ***suspendmanager*:**
  - take in account already built blocking buildings
  - don't consider tree branches as a suitable access path to a building

Misc Improvements

- Dreamfort: give noble suites double-thick walls and add apartment doors
- Suppress DF keyboard events when a DFHack keybinding is matched. This prevents, for example, a backtick from appearing in a textbox as text when you launch *gui/launcher* from the backtick keybinding.
- *autonick*: add more variety to nicknames based on famous literary dwarves
- *gui/unit-syndromes*: make lists searchable
- *logistics*: bring an autotraded bin to the depot if any item inside is tradeable instead of marking all items within the bin as untradeable if any individual item is untradeable
- *quickfort*: blueprint libraries are now moddable – add a `blueprints/` directory to your mod and they'll show up in *quickfort* and *gui/quickfort*!
- *stockpiles*: include exotic pets in the "tameable" filter
- *suspendmanager*: display the suspension reason when viewing a suspended building
- `widgets.EditField`: DFHack edit fields now support cut/copy/paste with the system clipboard with Ctrl-X/Ctrl-C/Ctrl-V

API

- `Items::markForTrade()`, `Items::isRequestedTradeGood()`, `Items::getValue`: see Lua notes below
- `Units::getUnitByNobleRole`, `Units::getUnitsByNobleRole`: unit lookup API by role

Internals

- Price calculations fixed for many item types

Lua

- `dfhack.items.getValue`: gained optional `caravan` and `caravan_buying` parameters for prices that take trader races and agreements into account
- `dfhack.items.isRequestedTradeGood`: discover whether an item is named in a trade agreement with an active caravan
- `dfhack.items.markForTrade`: mark items for trade
- `dfhack.units.getUnitByNobleRole`, `dfhack.units.getUnitsByNobleRole`: unit lookup API by role
- `widgets.TextButton`: wraps a `HotkeyLabel` and decorates it to look more like a button

Structures

- `build_req_choicest`: realign structure and fix vmethods
- `squad_orderst`: fix vmethods

Documentation

- *misery*: rewrite the documentation to clarify the actual effects of the plugin

7.12.20 DFHack 50.09-r1

Misc Improvements

- *caravan*: new overlay for selecting all/none on trade request screen
- *suspendmanager*: don't suspend constructions that are built over open space

Internals

- Core: update SDL interface from SDL1 to SDL2

Structures

- `tiletype_shape`: changed RAMP_TOP and ENDLESS_PIT to not walkable to reflect how scripts actually need these types to be treated

7.12.21 DFHack 50.08-r4

New Plugins

- *logistics*: automatically mark and route items or animals that come to monitored stockpiles. options are toggleable on an overlay that comes up when you have a stockpile selected.

Fixes

- *buildingplan*: don't include artifacts when max quality is masterful
- *dig-now*: clear item occupancy flags for channeled tiles that had items on them
- *emigration*: reassign home site for emigrating units so they don't just come right back to the fort
- *gui/create-item*: allow blocks to be made out of wood when using the restrictive filters
- *gui/liquids*: ensure tile temperature is set correctly when painting water or magma
- ***gui/quickfort***:
  - allow traffic designations to be applied over buildings
  - protect against meta blueprints recursing infinitely if they include themselves
- *gui/sandbox*: allow creatures that have separate caste-based graphics to be spawned (like ewes/rams)
- *RemoteFortressReader*: fix a crash with engravings with undefined images
- *workorder*: prevent `autoMilkCreature` from over-counting milkable animals, which was leading to cancellation spam for the MilkCreature job

Misc Improvements

- **Blueprint library:**
  - dreamfort: full rewrite and update for DF v50
  - pump_stack: updated walkthrough and separated dig and channel steps so boulders can be cleared
  - aquifer_tap: updated walkthrough
- *autonick*: additional nicknames based on burrowing animals, colours, gems, and minerals
- *combine*: reduce max different stacks in containers to 30 to prevent contaners from getting overfull
- *dig-now*: can now handle digging obsidian that has been formed from magma and water
- *gui/autodump*: add option to clear the `trader` flag from teleported items, allowing you to reclaim items dropped by merchants
- ***gui/control-panel***:
  - add some popular startup configuration commands for *autobutcher* and *autofarm*
  - add option for running *fix/blood-del* on new forts (enabled by default)

- ***gui/quickfort*:**
  - adapt "cursor lock" to mouse controls so it's easier to see the full preview for multi-level blueprints before you apply them
  - only display post-blueprint messages once when repeating the blueprint up or down z-levels
- *gui/sandbox*: when creating citizens, give them names appropriate for their races
- ***orders*:**
  - only display import/export/sort/clear panel on main orders screen
  - refine order conditions for library orders to reduce cancellation spam
- *prioritize*: add wild animal management tasks and lever pulling to the default list of prioritized job types
- *quickfort*: significant rewrite for DF v50! now handles zones, locations, stockpile configuration, hauling routes, and more
- *stockpiles*: added `barrels`, `organic`, `artifacts`, and `masterworks` stockpile presets
- ***suspendmanager*:**
  - now suspends construction jobs on top of floor designations, protecting the designations from being erased
  - suspend blocking jobs when building high walls or filling corridors
- *workorder*: reduce existing orders for automatic shearing and milking jobs when animals are no longer available

Removed

- *gui/automelt*: replaced by an overlay panel that appears when you click on a stockpile

Structures

- `abstract_building_libraryst`: initialize unknown variables as DF does
- `misc_trait_type`: realign

Documentation

- *Quickfort blueprint library*: update Dreamfort screenshots and links, add `aquifer_tap` screenshot

7.12.22 DFHack 50.08-r3

Fixes

- Fix crash for some players when they launch DF outside of the Steam client

7.12.23 DFHack 50.08-r2

New Plugins

- *add-spatter*: (reinstated) allow mods to add poisons and magical effects to weapons
- *changeitem*: (reinstated) change item material, quality, and subtype
- *createitem*: (reinstated) create arbitrary items from the command line
- *deramp*: (reinstated) removes all ramps designated for removal from the map
- *flows*: (reinstated) counts map blocks with flowing liquids
- *lair*: (reinstated) mark the map as a monster lair (this avoids item scatter when the fortress is abandoned)
- *luasocket*: (reinstated) provides a Lua API for accessing network sockets
- *work-now*: (reinstated, renamed from `workNow`) prevent dwarves from wandering aimlessly with "No job" after completing a task

New Scripts

- *assign-minecarts*: (reinstated) quickly assign minecarts to hauling routes
- *diplomacy*: view or alter diplomatic relationships
- *exportlegends*: (reinstated) export extended legends information for external browsing
- *fix/stuck-instruments*: fix instruments that are attached to invalid jobs, making them unusable. turn on automatic fixing in *gui/control-panel* in the `Maintenance` tab.
- *gui/autodump*: point and click item teleportation and destruction interface (available only if `armok` tools are shown)
- *gui/mod-manager*: automatically restore your list of active mods when generating new worlds
- *gui/sandbox*: creation interface for units, trees, and items (available only if `armok` tools are shown)
- *light-aquifers-only*: (reinstated) convert heavy aquifers to light
- *modtools/create-item*: (reinstated) commandline and API interface for creating items
- *necronomicon*: search fort for items containing the secrets of life and death

Fixes

- DFHack screen backgrounds now use appropriate tiles in DF Classic
- RemoteServer: fix crash on malformed json in `dfhack-config/remote-server.json`
- *autolabor*: work detail override warning now only appears on the work details screen
- *deathcause*: fix incorrect weapon sometimes being reported
- *gui/create-item*: allow armor to be made out of leather when using the restrictive filters
- *gui/design*: Fix building and stairs designation
- ***quickfort*:**
  - properly allow dwarves to smooth, engrave, and carve beneath walkable tiles of buildings

  - fixed detection of tiles where machines are allowed (e.g. water wheels *can* be built on stairs if there is a machine support nearby)
  - fixed rotation of blueprints with carved track tiles
- *RemoteFortressReader*: ensured names are transmitted in UTF-8 instead of CP437

Misc Improvements

- Core: new commandline flag/environment var: pass `--disable-dfhack` on the Dwarf Fortress commandline or specify `DFHACK_DISABLE=1` in the environment to disable DFHack for the current session.
- Dreamfort: improve traffic patterns throughout the fortress
- Settings: recover gracefully when settings files become corrupted (e.g. by DF CTD)
- **Window behavior:**
  - non-resizable windows now allow dragging by their frame edges by default
  - if you have multiple DFHack tool windows open, scrolling the mouse wheel while over an unfocused window will focus it and raise it to the top
- *autodump*: no longer checks for a keyboard cursor before executing, so `autodump destroy` (which doesn't require a cursor) can still function
- *gui/autodump*: fort-mode keybinding: Ctrl-H (when `armok` tools are enabled in *gui/control-panel*)
- *gui/blueprint*: recording of stockpile layouts and categories is now supported. note that detailed stockpile configurations will *not* be saved (yet)
- *gui/control-panel*: new preference for whether filters in lists search for substrings in the middle of words (e.g. if set to true, then "ee" will match "steel")
- *gui/create-item*: ask for number of items to spawn by default
- *gui/design*: Improved performance for drawing shapes
- ***gui/gm-editor*:**
  - when passing the `--freeze` option, further ensure that the game is frozen by halting all rendering (other than for DFHack tool windows)
  - Alt-A now enables auto-update mode, where you can watch values change live when the game is unpaused
- ***gui/quickfort*:**
  - blueprints that designate items for dumping/forbidding/etc. no longer show an error highlight for tiles that have no items on them
  - place (stockpile layout) mode is now supported. note that detailed stockpile configurations were part of query mode and are not yet supported
  - you can now generate manager orders for items required to complete bluerpints
- *light-aquifers-only*: now available as a fort Autostart option in *gui/control-panel*. note that it will only appear if "armok" tools are configured to be shown on the Preferences tab.
- *orders*: update orders in library for prepared meals, bins, archer uniforms, and weapons
- *overlay*: add links to the quickstart guide and the control panel on the DF title screen
- *stockpiles*: allow filtering creatures by tameability

Removed

- *orders*: `library/military_include_artifact_materials` library file removed since recent research indicates that platinum blunt weapons and silver crossbows are not more effective than standard steel. the alternate military orders file was also causing unneeded confusion.

Internals

- `dfhack.internal`: added memory analysis functions: `msizeAddress`, `getHeapState`, `heapTakeSnapshot`, `isAddressInHeap`, `isAddressActiveInHeap`, `isAddressUsedAfterFreeInHeap`, `getAddressSizeInHeap`, and `getRootAddressOfHeapObject`

Lua

- `ensure_keys`: walks a series of keys, creating new tables for any missing values
- `gui`: changed frame naming scheme to `FRAME_X` rather than `X_FRAME`, and added aliases for backwards compatibility. (for example `BOLD_FRAME` is now called `FRAME_BOLD`)
- `overlay.reload()`: has been renamed to `overlay.rescan()` so as not to conflict with the global `reload()` function. If you are developing an overlay, please take note of the new function name for reloading your overlay during development.

Structures

- Removed `steam_mod_manager` and `game_extra` globals. Their contents have been merged back into `game`.
- `abstract_building_contents`: identify fields and flags related to location item counts
- `arena_tree`: identify fields related to tree creation
- `arena_unit`: identify fields related to unit creation
- `mod_headerst`: rename `non_vanilla` flag to `vanilla` to reflect its actual usage
- `profession`: renamed captions `Cheese Maker` to `Cheesemaker`, `Bee Keeper` to `Beekeeper`, and `Bone Setter` to `Bone Doctor`

7.12.24 DFHack 50.08-r1

Fixes

- *autoclothing*: eliminate game lag when there are many inventory items in the fort
- ***buildingplan*:**
  - fixed size limit calculations for rollers
  - fixed items not being checked for accessibility in the filter and item selection dialogs
- *deteriorate*: ensure remains of enemy dwarves are properly deteriorated
- *dig-now*: properly detect and complete smoothing designations that have been converted into active jobs
- *suspendmanager*: Fix over-aggressive suspension of jobs that could still possibly be done (e.g. jobs that are partially submerged in water)

Misc Improvements

- ***buildingplan*:**
  - planner panel is minimized by default and now remembers minimized state
  - can now filter by gems (for gem windows) and yarn (for ropes in wells)
- *combine*: Now supports ammo, parts, powders, and seeds, and combines into containers
- *deteriorate*: add option to exclude useable parts from deterioration
- ***gui/control-panel*:**
  - add preference option for hiding the terminal console on startup
  - add preference option for hiding "armok" tools in command lists
- ***gui/gm-editor*:**
  - press `g` to move the map to the currently selected item/unit/building
  - press `Ctrl-D` to toggle read-only mode to protect from accidental changes; this state persists across sessions
  - new `--freeze` option for ensuring the game doesn't change while you're inspecting it
- *gui/launcher*: DFHack version now shown in the default help text
- *gui/prerelease-warning*: widgets are now clickable
- *overlay*: add the DFHack version string to the DF title screen
- `Dwarf Therapist`: add a warning to the Labors screen when Dwarf Therapist is active so players know that changes they make to that screen will have no effect. If you're starting a new embark and nobody seems to be doing anything, check your Labors tab for this warning to see if Dwarf Therapist thinks it is in control (even if it's not running).
- `toggle-kbd-cursor`: add hotkey for toggling the keyboard cursor (Alt-K)
- `version`: add alias to display the DFHack help (including the version number) so something happens when players try to run "version"

Removed

- *title-version*: replaced by an *overlay* widget

Lua

- `gui.ZScreenModal`: ZScreen subclass for modal dialogs
- `widgets.CycleHotkeyLabel`: exposed "key_sep" and "option_gap" attributes for improved stylistic control.
- `widgets.RangeSlider`: new mouse-controlled two-headed slider widget

Structures

- convert `mod_manager` fields to pointers

7.12.25 DFHack 50.07-r1

New Plugins

- *faststart*: speeds up the "Loading…" screen so the Main Menu appears faster

Fixes

- *blueprint*: interpret saplings, shrubs, and twigs as floors instead of walls
- *caravan*: fix trade good list sometimes disappearing when you collapse a bin
- *combine*: fix error processing stockpiles with boundaries that extend outside of the map
- *gui/control-panel*: the config UI for *automelt* is no longer offered when not in fortress mode
- *gui/gm-editor*: no longer nudges last open window when opening a new one
- *hotkeys*: hotkey hints on menu popup will no longer get their last character cut off by the scrollbar
- *prospector*: display both "raw" Z levels and "cooked" elevations
- ***stockpiles*:**
  - fix crash when importing settings for gems from other worlds
  - allow numbers in saved stockpile filenames
- *warn-starving*: no longer warns for dead units
- `launchdf`: launch Dwarf Fortress via the Steam client so Steam Workshop is functional

Misc Improvements

- Core: hide DFHack terminal console by default when running on Steam Deck
- **Mods:**
  - scripts in mods that are only in the steam workshop directory are now accessible. this means that a script-only mod that you never mark as "active" when generating a world will still receive automatic updates and be usable from in-game
  - scripts from only the most recent version of an installed mod are added to the script path
  - give active mods a chance to reattach their load hooks when a world is reloaded
- ***buildingplan*:**
  - items in the item selection dialog should now use the same item quality symbols as the base game
  - hide planner overlay while the DF tutorial is active so that it can detect when you have placed the carpenter's workshop and bed and allow you to finish the tutorial
  - can now filter by cloth and silk materials (for ropes)
  - rearranged elements of `planneroverlay` interface
  - rearranged elements of `itemselection` interface

- ***gui/control-panel*:**
  - bugfix services are now enabled by default
  - add *faststart* to the system services
- ***gui/gm-editor*:**
  - can now jump to material info objects from a mat_type reference with a mat_index using `i`
  - the key column now auto-fits to the widest key
- *prioritize*: revise and simplify the default list of prioritized jobs – be sure to tell us if your forts are running noticeably better (or worse!)

Lua

- added two new window borders: `gui.BOLD_FRAME` for accented elements and `gui.INTERIOR_MEDIUM_FRAME` for a signature-less frame that's thicker than the existing `gui.INTERIOR_FRAME`

Structures

- correct bit size of tree body data
- identified fields in `deep_vein_hollow`, `glowing_barrier`, and `cursed_tomb` map events
- identified `divine_treasure` and `encased_horror` map events

Documentation

- *Installing*: updated to include Steam installation instructions

7.12.26 DFHack 50.07-beta2

New Plugins

- *getplants*: reinstated: designate trees for chopping and shrubs for gathering according to type
- *prospector*: reinstated: get stone, ore, gem, and other tile property counts in fort mode.

New Scripts

- *fix/general-strike*: fix known causes of the general strike bug (contributed by Putnam)
- *gui/civ-alert*: configure and trigger civilian alerts
- *gui/seedwatch*: GUI config and status panel interface for *seedwatch*

Fixes

- ***buildingplan***:
  - filters are now properly applied to planned stairs
  - existing carved up/down stairs are now taken into account when determining which stair shape to construct
  - upright spike traps are now placed extended rather than retracted
  - you can no longer designate constructions on tiles with magma or deep water, mirroring the vanilla restrictions
  - fixed material filters getting lost for planning buildings on save/reload
  - respect building size limits (e.g. roads and bridges cannot be more than 31 tiles in any dimension)
- *caravan*: item list length now correct when expanding and collapsing containers
- *prioritize*: fixed all watched job type names showing as `nil` after a game load
- ***suspendmanager***:
  - does not suspend non-blocking jobs such as floor bars or bridges anymore
  - fix occasional bad identification of buildingplan jobs
- ***tailor***:
  - properly discriminate between dyed and undyed cloth
  - no longer default to using adamantine cloth for producing clothes
  - take queued orders into account when calculating available materials
  - skip units who can't wear clothes
  - identify more available items as available, solving issues with over-production
- *warn-starving*: no longer warns for enemy and neutral units

Misc Improvements

- Mods: scripts in mods are now automatically added to the DFHack script path. DFHack recognizes two directories in a mod's folder: `scripts_modinstalled/` and `scripts_modactive/`. `scripts_modinstalled/` folders will always be added the script path, regardless of whether the mod is active in a world. `scripts_modactive/` folders will only be added to the script path when the mod is active in the current loaded world.
- *automelt*: now allows metal chests to be melted (workaround for DF bug 2493 is no longer needed)
- ***buildingplan***:
  - filters and global settings are now ignored when manually choosing items for a building, allowing you to make custom choices independently of the filters that would otherwise be used
  - if *suspendmanager* is running, then planned buildings will be left suspended when their items are all attached. *suspendmanager* will unsuspsend them for construction when it is safe to do so.
  - add option for autoselecting the last manually chosen item (like *automaterial* used to do)
- ***combine***:
  - you can select a target stockpile in the UI instead of having to use the keyboard cursor

  - added `--quiet` option for no output when there are no changes
- *confirm*: adds confirmation for removing burrows via the repaint menu
- *enable*: can now interpret aliases defined with the *alias* command
- *exterminate*: add support for `vaporize` kill method for when you don't want to leave a corpse
- ***gui/control-panel*:**
  - Now detects overlays from scripts named with capital letters
  - added `combine all` maintenance option for automatic combining of partial stacks in stockpiles
  - added `general-strike` maintenance option for automatic fixing of (at least one cause of) the general strike bug
- ***gui/cp437-table*:**
  - now has larger key buttons and clickable backspace/submit/cancel buttons, making it fully usable on the Steam Deck and other systems that don't have an accessible keyboard
  - dialog is now fully controllable with the mouse, including highlighting which key you are hovering over and adding a clickable backspace button
- *gui/design*: Now supports placing constructions using 'Building' mode. Inner and Outer tile constructions are configurable. Uses buildingplan filters set up with the regular buildingplan interface.
- ***orders*:**
  - add minimize button to overlay panel so you can get it out of the way to read long statue descriptions when choosing a subject in the details screen
  - add option to delete exported files from the import dialog
- ***stockpiles*:**
  - support applying stockpile configurations with fully enabled categories to stockpiles in worlds other than the one where the configuration was exported from
  - support partial application of a saved config based on dynamic filtering (e.g. disable all tallow in a food stockpile, even tallow from world-specific generated creatures)
  - additive and subtractive modes when applying a second stockpile configuration on top of a first
  - write player-exported stockpile configurations to the `dfhack-config/stockpiles` folder. If you have any stockpile configs in other directories, please move them to that folder.
  - now includes a library of useful stockpile configs (see docs for details)
- ***stripcaged*:**
  - added `--skip-forbidden` option for greater control over which items are marked for dumping
  - items that are marked for dumping are now automatically unforbidden (unless `--skip-forbidden` is set)

Removed

- *autounsuspend*: replaced by *suspendmanager*
- *gui/dig*: renamed to *gui/design*

Lua

- **`widgets.CycleHotkeyLabel`:**
  - options that are bare integers will no longer be interpreted as the pen color in addition to being the label and value
  - option labels and pens can now be functions that return a label or pen
- **`widgets.Label`:**
  - tokens can now specify a `htile` property to indicate the tile that should be shown when the Label is hovered over with the mouse
  - click handlers no longer get the label itself as the first param to the click handler

Structures

- realigned `furniture_type` enum (added BAG)
- realigned `stockpile_settings` for new "corpses" vector

Documentation

- the `untested` tag has been renamed to `unavailable` to better reflect the status of the remaining unavaialable tools. most of the simply "untested" tools have now been tested and marked as working. the remaining tools are known to need development work before they are available again.
- *DFHack modding guide*: guide updated to include information for 3rd party script developers

7.12.27 DFHack 50.07-beta1

New Scripts

- *gui/suspendmanager*: graphical configuration interface for *suspendmanager*
- *suspendmanager*: automatic job suspension management (replaces *autounsuspend*)
- *suspend*: suspends building construction jobs

Fixes

- ***buildingplan*:**
  - items are now attached correctly to screw pumps and other multi-item buildings
  - buildings with different material filters will no longer get "stuck" if one of the filters currently matches no items
- *gui/launcher*: tab characters in command output now appear as a space instead of a code page 437 "blob"
- *quicksave*: now reliably triggers an autosave, even if one has been performed recently
- *showmood* properly count required number of bars and cloth when they aren't the main item for the strange mood

Misc Improvements

- ***Quickfort blueprint library*:**
  - library blueprints have moved from `blueprints` to `hack/data/blueprints`
  - player-created blueprints should now go in the `dfhack-config/blueprints` folder. please move your existing blueprints from `blueprints` to `dfhack-config/blueprints`. you don't need to move the library blueprints – those can be safely deleted from the old `blueprints` directory.
- *blueprint*: now writes blueprints to the `dfhack-config/blueprints` directory
- ***buildingplan*:**
  - can now filter by clay materials
  - remember choice per building type for whether the player wants to choose specific items
  - you can now attach multiple weapons to spike traps
  - can now filter by whether a slab is engraved
  - add "minimize" button to temporarily get the planner overlay out of the way if you would rather use the vanilla UI for placing the current building
  - add `buildingplan reset` command for resetting all filters to defaults
  - rename "Build" button to "Confirm" on the item selection dialog and change the hotkey from "B" to "C"
- *gui/gm-editor*: can now open the selected stockpile if run without parameters
- *quickfort*: now reads player-created blueprints from `dfhack-config/blueprints/` instead of the old `blueprints/` directory. Be sure to move over your personal blueprints to the new directory!
- *showmood*: clarify how many bars and/or cloth items are actually needed for the mood

Removed

- *buildingplan*: "heat safety" setting is temporarily removed while we investigate incorrect item matching

Structures

- identified two fields related to saves/autosaves to facilitate quicksave implementation

7.12.28 DFHack 50.07-alpha3

Fixes

- *dig-now*: fixed multi-layer channel designations only channeling every second layer
- *gui/create-item*: fix generic corpsepiece spawning
- `dfhack.job.isSuitableMaterial`: now properly detects lack of fire and magma safety for vulnerable materials with high melting points
- `widgets.HotkeyLabel`: don't trigger on click if the widget is disabled

Misc Improvements

- *buildingplan*: entirely new UI for building placement, item selection, and materials filtering!
- *dig-now*: added handling of dig designations that have been converted into active jobs
- *gui/create-item*: added ability to spawn 'whole' corpsepieces (every layer of a part)
- ***gui/dig*:**
  - Allow placing an extra point (curve) while still placing the second main point
  - Allow placing n-point shapes, shape rotation/mirroring
  - Allow second bezier point, mirror-mode for freeform shapes, symmetry mode

Removed

- *automaterial*: all functionality has been merged into *buildingplan*
- `gui.THIN_FRAME`: replaced by `gui.INTERIOR_FRAME`

API

- Gui focus strings will no longer get the "dfhack/" prefix if the string "dfhack/" already exists in the focus string
- `Maps::GetBiomeTypeRef` renamed to `Maps::getBiomeTypeRef` for consistency
- `Maps::GetBiomeType` renamed to `Maps::getBiomeType` for consistency
- **`Military`:**
  - New module for military functionality
  - new `makeSquad` to create a squad
  - changed `getSquadName` to take a squad identifier
  - new `updateRoomAssignments` for assigning a squad to a barracks and archery range

Lua

- `dfhack.job.attachJobItem()`: allows you to attach specific items to a job
- `dfhack.screen.paintTile()`: you can now explicitly clear the interface cursor from a map tile by passing `0` as the tile value
- `gui.INTERIOR_FRAME`: a panel frame style for use in highlighting off interior areas of a UI
- `maps.getBiomeType`: exposed preexisting function to Lua
- `widgets.CycleHotkeyLabel`: add `label_below` attribute for compact 2-line output
- `widgets.FilteredList`: search key matching is now case insensitive by default
- `widgets.Label`: token `tile` properties can now be functions that return a value

Structures

- `history_eventst`: Removed `history_event_masterpiece_created_arch_designst` and related enum value
- `plot_infost.``unk_8``: renamed to `theft_intrigues`. Fields `unk_1` thru `unk_8` renamed to `target_item`, `mastermind_hf`, `mastermind_plot_id`, `corruptor_hf`, `corruptor`, `corruptee_hf`, `corruptee`, and `theft_agreement`. `unk_1` renamed to `item_known_pos`.
- `specific_ref_type`: Removed `BUILDING_PARTY`, `PETINFO_PET`, and `PETINFO_OWNER` enum values to fix alignment.

7.12.29 DFHack 50.07-alpha2

New Scripts

- *combine*: combines stacks of food and plant items.

Fixes

- *autobutcher*: implemented work-around for Dwarf Fortress not setting nicknames properly, so that nicknames created in the in-game interface are detected & protect animals from being butchered properly. Note that nicknames for unnamed units are not currently saved by dwarf fortress - use `enable fix/protect-nicks` to fix any nicknames created/removed within dwarf fortress so they can be saved/reloaded when you reload the game.
- *autochop*: generate default names for burrows with no assigned names
- *autodump*: changed behaviour to only change `dump` and `forbid` flags if an item is successfully dumped.
- *channel-safely*: fix an out of bounds error regarding the REPORT event listener receiving (presumably) stale id's
- *confirm*: fix fps drop when enabled
- *devel/query*: can now properly index vectors in the –table argument
- *forbid*: fix detection of unreachable items for items in containers
- *gui/blueprint*: correctly use setting presets passed on the commandline
- *gui/quickfort*: correctly use settings presets passed on the commandline
- *makeown*: fixes errors caused by using makeown on an invader

- *nestboxes*: fixed bug causing nestboxes themselves to be forbidden, which prevented citizens from using them to lay eggs. Now only eggs are forbidden.
- *seedwatch*: fix saving and loading of seed stock targets
- *tailor*: block making clothing sized for toads; make replacement clothing orders use the size of the wearer, not the size of the garment
- *troubleshoot-item*: fix printing of job details for chosen item
- *unforbid*: fix detection of unreachable items for items in containers
- `Buildings::StockpileIterator`: fix check for stockpile items on block boundary.

Misc Improvements

- DFHack tool windows that capture mouse clicks (and therefore prevent you from clicking on the "pause" button) now unconditionally pause the game when they open (but you can still unpause with the keyboard if you want to). Examples of this behavior: *gui/quickfort*, *gui/blueprint*, *gui/liquids*
- Stopped mouse clicks from affecting the map when a click on a DFHack screen dismisses the window
- *autobutcher*: logs activity to the console terminal instead of making disruptive in-game announcements
- *caravan*: add trade screen overlay that assists with seleting groups of items and collapsing groups in the UI
- *confirm*: configuration data is now persisted globally.
- *devel/query*: will now search for jobs at the map coordinate highlighted, if no explicit job is highlighted and there is a map tile highlighted
- *devel/visualize-structure*: now automatically inspects the contents of most pointer fields, rather than inspecting the pointers themselves
- *gui/gm-editor*: will now inspect a selected building itself if the building has no current jobs
- *showmood*: now shows the number of items needed for cloth and bars in addition to the technically correct but always confusing "total dimension" (150 per bar or 10,000 per cloth)
- *tailor*: add support for adamantine cloth (off by default); improve logging
- ***troubleshoot-item*:**
  - output as bullet point list with indenting, with item description and ID at top
  - reports on items that are hidden, artifacts, in containers, and held by a unit
  - reports on the contents of containers with counts for each contained item type

Removed

- *combine-drinks*: replaced by *combine*
- *combine-plants*: replaced by *combine*

API

- Units module: added new predicates for `isGeldable()`, `isMarkedForGelding()`, and `isPet()`
- `Gui::any_civzone_hotkey`, `Gui::getAnyCivZone`, `Gui::getSelectedCivZone`: new functions to operate on the new zone system

Lua

- `dfhack.gui.getSelectedCivZone`: returns the Zone that the user has selected currently
- `widgets.FilteredList`: Added `edit_on_change` optional parameter to allow a custom callback on filter edit change.
- `widgets.TabBar`: new library widget (migrated from control-panel.lua)

Structures

- corrected alignment in `world.status`
- identify item vmethod 213 (applies a thread improvements to appropriate items based on an RNG)
- identify two anons in `difficultyst`
- identify various data types related to job completion/cancellation
- split `gamest` into `gamest` and `gamest_extra` to accommodate steam-specific data in `gamest.mod_manager`
- `activity_info`: `unit_actor`, `unit_noble`, and `place` converted from pointers to integer references.
- `dipscript_popup`: `meeting_holder` converted from unit pointer into two unit refs `meeting_holder_actor` and `meeting_holder_noble`.
- `plotinfost.``equipment``: Converted `items_unmanifested`, `items_unassigned`, and `items_assigned` vectors from pointers to item refs

7.12.30 DFHack 50.07-alpha1

New Scripts

- *gui/design*: digging and construction designation tool with shapes and patterns
- *makeown*: makes the selected unit a citizen of your fortress

Fixes

- Fix persisted data not being written on manual save
- Fix right click sometimes closing both a DFHack window and a vanilla panel
- Fixed issue with scrollable lists having some data off-screen if they were scrolled before being made visible
- *autochop*: fixed bug related to lua stack smashing behavior in returned stockpile configs
- *automelt*: fixed bug related to lua stack smashing behavior in returned stockpile configs
- *channel-safely*: fixed bug resulting in marker mode never being set for any designation
- *fix/protect-nicks*: now works by setting the historical figure nickname

- *gui/dig*: Fix for 'continuing' auto-stair designation. Avoid nil index issue for tile_type
- *gui/liquids*: fixed issues with unit pathing after adding/removing liquids
- *gui/unit-syndromes*: allow the window widgets to be interacted with
- ***nestboxes***:
  - now cancels any in-progress hauling jobs when it protects a fertile egg
  - now scans for eggs more frequently and cancels any in-progress hauling jobs when it protects a fertile egg
- `Units::isFortControlled`: Account for agitated wildlife

Misc Improvements

- replaced DFHack logo used for the hover hotspot with a crisper image
- ***autobutcher***:
  - changed defaults from 5 females / 1 male to 4 females / 2 males so a single unfortunate accident doesn't leave players without a mating pair
  - now immediately loads races available at game start into the watchlist
- ***autodump***:
  - reinstate `autodump-destroy-item`, hotkey: Ctrl-K
  - new hotkey for `autodump-destroy-here`: Ctrl-H
- *automelt*: is now more resistent to vanilla savegame corruption
- *cleaners*: new hotkey for *cleaners*: Ctrl-C
- *dig*: new hotkeys for vein designation on z-level (Ctrl-V) and vein designation across z-levels (Ctrl-Shift-V)
- *gui/dig* : Added 'Line' shape that also can draw curves, added draggable center handle
- ***gui/gm-editor***:
  - now supports multiple independent data inspection windows
  - now prints out contents of coordinate vars instead of just the type
- *hotkeys*: DFHack logo is now hidden on screens where it covers important information when in the default position (e.g. when choosing an embark site)
- *misery*: now persists state with the fort
- *orders*: recipe for silver crossbows removed from `library/military` as it is not a vanilla recipe, but is available in `library/military_include_artifact_materials`
- *rejuvenate*: now takes an –age parameter to choose a desired age.
- *stonesense*: added an `INVERT_MOUSE_Z` option to invert the mouse wheel direction

Lua

- *overlay*: overlay widgets can now specify focus paths for the viewscreens they attach to so they only appear in specific contexts. see *DFHack overlay dev guide* for details.
- `widgets.CycleHotkeyLabel`: Added `key_back` optional parameter to cycle backwards.
- `widgets.FilteredList`: Added `case_sensitive` optional paramter to determine if filtering is case sensitive.
- **`widgets.HotkeyLabel:`**
  - Added `setLabel` method to allow easily updating the label text without mangling the keyboard shortcut.
  - Added `setOnActivate` method to allow easily updating the `on_activate` callback.

Structures

- added missing tiletypes and corrected a few old ones based on a list supplied by Toady

7.12.31 DFHack 50.05-alpha3.1

Fixes

- *gui/launcher*: no longer resets to the Help tab on every keystroke
- *seedwatch*: fix parameter parsing when setting targets

7.12.32 DFHack 50.05-alpha3

New Plugins

- *autoslab*: automatically create work orders to engrave slabs for ghostly dwarves

New Scripts

- *autofish*: auto-manage fishing labors to control your stock of fish
- *fix/civil-war*: removes negative relations with own government
- *fix/protect-nicks*: restore nicknames when DF loses them
- *forbid*: forbid and list forbidden items on the map
- *gui/autofish*: GUI config and status panel interface for autofish
- *gui/automelt*: GUI config and status panel interface for automelt
- *gui/control-panel*: quick access to DFHack configuration
- *gui/unit-syndromes*: browser for syndrome information

Fixes

- allow launcher tools to launch themselves without hanging the game
- DF screens can no longer get "stuck" on transitions when DFHack tool windows are visible. Instead, those DF screens are force-paused while DFHack windows are visible so the player can close them first and not corrupt the screen sequence. The "PAUSE FORCED" indicator will appear on these DFHack windows to indicate what is happening.
- fix issues with clicks "passing through" some DFHack window elements to the screen below
- *autochop*: fixed a crash when processing trees with corrupt data structures (e.g. when a trunk tile fails to fall when the rest of the tree is chopped down)
- *autoclothing*: fixed a crash that can happen when units are holding invalid items.
- *build-now*: now correctly avoids adjusting non-empty tiles above constructions that it builds
- *catsplosion*: now only affects live, active units
- *getplants*: trees are now designated correctly
- ***orders*:**
  - fix orders in library/basic that create bags
  - library/military now sticks to vanilla rules and does not add orders for normally-mood-only platinum weapons. A new library orders file `library/military_include_artifact_materials` is now offered as an alternate `library/military` set of orders that still includes the platinum weapons.
- *quickfort*: allow floor bars, floor grates, and hatches to be placed over all stair types like vanilla allows

Misc Improvements

- DFHack windows can now be "defocused" by clicking somewhere not over the tool window. This has the same effect as pinning previously did, but without the extra clicking.
- New borders for DFHack tool windows – tell us what you think!
- Windows now display "PAUSE FORCED" on the lower border if the tool is forcing the game to pause
- *autoclothing*: merged the two separate reports into the same command.
- *automelt*: stockpile configuration can now be set from the commandline
- ***ban-cooking*:**
  - ban announcements are now hidden by default; use new option `--verbose` to show them.
  - report number of items banned.
- *build-now*: now handles dirt roads and initializes farm plots properly
- *channel-safely*: new monitoring for cave-in prevention
- *devel/click-monitor*: report on middle mouse button actions
- *getplants*: ID values will now be accepted regardless of case
- *gui/autochop*: hide uninteresting burrows by default
- *gui/blueprint*: allow map movement with the keyboard while the UI is open
- ***gui/control-panel*:**
  - you can now configure whether DFHack tool windows should pause the game by default

  - new global hotkey for quick access: Ctrl-Shift-E

- ***gui/create-item*:**
  - support spawning corpse pieces (e.g. shells) under "body part"
  - added search and filter capabilities to the selection lists
  - added whole corpse spawning alongside corpsepieces. (under "corpse")
- ***gui/launcher*:**
  - sped up initialization time for faster window appearance
  - make command output scrollback separate from the help text so players can continue to see the output of the previous command as they type the next one
  - allow double spacebar to pause/unpause the game, even while typing a command
  - clarify what is being shown in the autocomplete list (all commands, autocompletion of partially typed command, or commands related to typed command)
  - support running commands directly from the autocomplete list via double-clicking
- *gui/liquids*: interface overhaul, also now allows spawning river sources, setting/adding/removing liquid levels, and cleaning water from being salty or stagnant
- *gui/overlay*: now focuses on repositioning overlay widgets; enabling, disabling, and getting help for overlay widgets has moved to the new *gui/control-panel*
- ***gui/quickcmd*:**
  - now has its own global keybinding for your convenience: Ctrl-Shift-A
  - now acts like a regular window instead of a modal dialog
- *gui/quickfort*: don't close the window when applying a blueprint so players can apply the same blueprint multiple times more easily
- *hotkeys*: clicking on the DFHack logo no longer closes the popup menu
- *locate-ore*: now only searches revealed tiles by default
- *modtools/spawn-liquid*: sets tile temperature to stable levels when spawning water or magma
- *nestboxes*: now saves enabled state in your savegame
- *orders*: orders plugin functionality is now accessible via an *overlay* widget when the manager orders screen is open
- ***prioritize*:**
  - pushing minecarts is now included in the default priortization list
  - now automatically starts boosting the default list of job types when enabled
- *quickfort*: planned buildings are now properly attached to any pertinent overlapping zones
- *seedwatch*: now persists enabled state in the savegame, automatically loads useful defaults, and respects reachability when counting available seeds
- *unforbid*: avoids unforbidding unreachable and underwater items by default

Removed

- *autohauler*: no plans to port to v50, as it just doesn't make sense with the new work detail system
- *show-unit-syndromes*: replaced by *gui/unit-syndromes*; html export is no longer supported

API

- *overlay*: overlay widgets can now specify a default enabled state if they are not already set in the player's overlay config file
- `Buildings::containsTile()`: no longer takes a `room` parameter since that's not how rooms work anymore. If the building has extents, the extents will be checked. otherwise, the result just depends on whether the tile is within the building's bounding box.
- `Lua::Push`: now supports `std::unordered_map`
- `Screen::Pen`: now accepts `top_of_text` and `bottom_of_text` properties to support offset text in graphics mode
- `Units::getCitizens()`: gets a list of citizens, which otherwise you'd have to iterate over all units the world to discover

Lua

- ***helpdb*:**
  - new function: `helpdb.refresh()` to force a refresh of the database. Call if you are a developer adding new scripts, loading new plugins, or changing help text during play
  - changed from auto-refreshing every 60 seconds to only refreshing on explicit call to `helpdb.refresh()`. docs very rarely change during a play session, and the automatic database refreshes were slowing down the startup of *gui/launcher* and anything else that displays help text.
- *tiletypes*: now has a Lua API! `tiletypes_setTile`
- `dfhack.units.getCitizens()`: gets a list of citizens
- `gui.ZScreen`: new attribute: `defocusable` for controlling whether a window loses keyboard focus when the map is clicked
- **`widgets.Label`:**
  - `label.scroll()` now understands `home` and `end` keywords for scrolling to the top or bottom
  - token `tile` properties can now be either pens or numeric texture ids
- `widgets.List`: new callbacks for double click and shift double click

Structures

- add "hospital" language name category
- corrected misalignment in `unitst` (affecting `occupation` and `adjective`)
- identified fields for squads and other military structures
- identified some anons in `unitst` related to textures (thanks, putnam)
- identify a table of daily events scheduled to take place inthe current year
- realigned and fleshed out `entity_site_link` (again, thanks, putnam)
- remove some no-longer-valid reputation types
- `building_design`: corrected misalignments
- `creature_raw_graphics`: corrected misalignments
- `item.setSharpness()`: more info about params
- `occupation_type`: add enum values for new occupations related to hospitals

Documentation

- Quickstart guide has been updated with info on new window behavior and how to use the control panel
- *Compilation*: instructions added for cross-compiling DFHack for Windows from a Linux Docker builder

7.12.33 DFHack 50.05-alpha2

Fixes

- *autofarm*: don't duplicate status line entries for crops with no current supply
- *build-now*: don't error on constructions that do not have an item attached
- *gui/gm-editor*: fix errors displayed while viewing help screen
- *orders*: allow the orders library to be listed and imported properly (if you previously copied the orders library into your `dfhack-config/orders` directory to work around this bug, you can remove those files now)
- *tailor*: now respects the setting of the "used dyed clothing" standing order toggle

Removed

- *create-items*: replaced by *gui/create-item* `--multi`

Structures

- corrected misalignment in `historical_entity`
- identified two old and one new language name groups
- partially identified squad-related structures in `plotinfo` and corrected position of `civ_alert_idx` (does not affect alignment)

7.12.34 DFHack 50.05-alpha1

New Scripts

- *allneeds*: list all unmet needs sorted by how many dwarves suffer from them.
- *devel/tile-browser*: page through available textures and see their texture ids
- *gui/autochop*: configuration frontend and status monitor for the *autochop* plugin

Fixes

- *make-legendary*: "MilitaryUnarmed" option now functional
- `widgets.WrappedLabel`: no longer resets scroll position when window is moved or resized

Misc Improvements

- Scrollable widgets now react to mouse wheel events when the mouse is over the widget
- the `dfhack-config/scripts/` folder is now searched for scripts by default
- *autounsuspend*: now saves its state with your fort
- *devel/inspect-screen*: updated for new rendering semantics and can now also inspect map textures
- *emigration*: now saves its state with your fort
- *exterminate*: added drown method. magma and drown methods will now clean up liquids automatically.
- *gui/cp437-table*: converted to a movable, mouse-enabled window
- *gui/gm-editor*: converted to a movable, resizable, mouse-enabled window
- *gui/gm-unit*: converted to a movable, resizable, mouse-enabled window
- ***gui/launcher*:**
  - now supports a smaller, minimal mode. click the toggle in the launcher UI or start in minimal mode via the `Ctrl-Shift-P` keybinding
  - can now be dragged from anywhere on the window body
  - now remembers its size and position between invocations
- ***gui/quickcmd*:**
  - converted to a movable, resizable, mouse-enabled window
  - commands are now stored globally so you don't have to recreate commands for every fort
- *hotkeys*: overlay hotspot widget now shows the DFHack logo in graphics mode and "DFHack" in text mode

- *prioritize*: now saves its state with your fort
- *Script paths*: removed "raw" directories from default script paths. now the default locations to search for scripts are `dfhack-config/scripts`, `save/*/scripts`, and `hack/scripts`
- ***unsuspend*:**
  - overlay now displays different letters for different suspend states so they can be differentiated in graphics mode (P=planning, x=suspended, X=repeatedly suspended)
  - overlay now shows a marker all the time when in graphics mode. ascii mode still only shows when paused so that you can see what's underneath.
- `init.d`: directories have moved from the `raw` subfolder (which no longer exists) to the root of the main DF folder or a savegame folder

Removed

- Ruby is no longer a supported DFHack scripting language
- `fix-job-postings` from the *workflow* plugin is now obsolete since affected savegames can no longer be loaded

API

- `Gui::getDFViewscreen`: returns the topmost underlying DF viewscreen
- `Gui::getDwarfmodeDims`: now only returns map viewport dimensions; menu dimensions are obsolete
- `Screen::Pen`: now accepts `keep_lower` and `write_to_lower` properties to support foreground and background textures in graphics mode

Lua

- Removed `os.execute()` and `io.popen()` built-in functions
- *overlay*: `OverlayWidget` now inherits from `Panel` instead of `Widget` to get all the frame and mouse integration goodies
- `dfhack.gui.getDFViewscreen()`: returns the topmost underlying DF viewscreen
- `gui.CLEAR_PEN`: now clears the background and foreground and writes to the background (before it would always write to the foreground)
- `gui.KEEP_LOWER_PEN`: a general use pen that writes associated tiles to the foreground while keeping the existing background
- **`gui.View`:**
  - `visible` and `active` can now be functions that return a boolean
  - new function `view:getMouseFramePos()` for detecting whether the mouse is within (or over) the exterior frame of a view
- `gui.ZScreen`: Screen subclass that implements window raising, multi-viewscreen input handling, and viewscreen event pass-through so the underlying map can be interacted with and dragged around while DFHack screens are visible
- **`widgets.CycleHotkeyLabel`:**
  - now supports rendering option labels in the color of your choice

    - new functions `setOption()` and `getOptionPen()`
- `widgets.Label`: tiles can now have an associated width
- `widgets.Panel`: new attributes to control window dragging and resizing with mouse or keyboard
- `widgets.ToggleHotkeyLabel`: now renders the `On` option in green text
- `widgets.Window`: Panel subclass with attributes preset for top-level windows

Structures

- **Renamed globals to match DF:**
    - `ui`: renamed to `plotinfo`
    - `ui_advmode`: renamed to `adventure`
    - `ui_build_selector`: renamed to `buildreq`
    - `ui_sidebar_menus`: renamed to `game`
- `building_civzonest`: identify two variables, `dir_x` and `dir_y`, that handle archery range direction.

Documentation

- added DFHack architecture diagrams to the dev intro
- added DFHack Quickstart guide
- *devel/hello-world*: updated to be a better example from which to start new gui scripts
- *DFHack overlay dev guide*: added troubleshooting tips and common development workflows

7.12.35 DFHack 0.47.05-r8

New Plugins

- *channel-safely*: auto-manage channel designations to keep dwarves safe
- *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
  - new action timer API for speeding up of slowing down units
- `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

7.12.36 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.

7.12.37 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 *deterioratecloth*es, *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
- ***The "Autostart" subtab***:
  - 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 blueprint creation guide*:**
  - 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

- *deterioriateclothes*: 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`

7.12.38 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`
- ***The "Autostart" subtab***:
  - 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)`

7.12.39 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*, *The "Autostart" subtab*: 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`

7.12.40 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

- *The "Autostart" subtab*: documentation for all of *Dreamfort*'s supporting files (useful for all forts, not just Dreamfort!)
- *Quickfort blueprint library*: updated dreamfort documentation and added screenshots

7.12.41 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*

7.12.42 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*

7.12.43 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

7.12.44 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

7.12.45 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 blueprint creation guide*: alias syntax and alias standard library documentation for *quickfort* blueprints
- *Quickfort blueprint library*: overview of the quickfort blueprint library

7.12.46 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

7.12.47 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

7.12.48 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

7.12.49 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

7.12.50 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)

7.12.51 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*)

7.12.52 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

7.12.53 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_civliststst`)
- `spoils_report`: new type, most fields identified
- **`viewscreen_civliststst`:**
  - split `unk_20` into 3 pointers
  - identified new pages
  - identified new messenger-related fields
- **`viewscreen_image_creatorst`:**
  - fixed layout
  - identified many fields
- `viewscreen_reportliststst`: added new mission and spoils report-related fields (fixed layout)
- `world.languages`: identified (minimal information; whole languages stored elsewhere)
- **`world.status`:**
  - `mission_reports`: renamed, was `missions`
  - `spoils_reports`: identified
- `world.unk_131ec0`, `world.unk_131ef0`: researched layout

- `world.worldgen_status`: identified many fields
- `world`: `belief_systems`: identified

7.12.54 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

7.12.55 DFHack 0.44.11-beta2.1

Internals

- *stonesense*: fixed build

7.12.56 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*

7.12.57 DFHack 0.44.11-beta1

Fixes

- Fixed displayed names (from `Units::getVisibleName`) for units with identities
- Fixed potential memory leak in `Screen::show()`
- *fix/dead-units*: fixed script trying to use missing isDiplomat function

Misc Improvements

- **Console:**
  - added support for multibyte characters on Linux/macOS
  - made the console exit properly when an interactive command is active (*liquids*, *mode*, *tiletypes*)
- Linux: added automatic support for GCC sanitizers in `dfhack` script
- Made the `DFHACK_PORT` environment variable take priority over `remote-server.json`
- *dfhack-run*: added support for port specified in `remote-server.json`, to match DFHack's behavior
- *digfort*: added better map bounds checking
- ***remove-stress*:**
  - added support for `-all` as an alternative to the existing `all` argument for consistency
  - sped up significantly
  - improved output/error messages
  - now removes tantrums, depression, and obliviousness
- *ruby*: sped up handling of onupdate events

API

- Exposed `Screen::zoom()` to C++ (was Lua-only)
- New functions: `Units::isDiplomat(unit)`

Internals

- jsoncpp: updated to version 1.8.4 and switched to using a git submodule

Lua

- Added `printall_recurse` to print tables and DF references recursively. It can be also used with `^` from the *lua* interpreter.
- `gui.widgets`: `List:setChoices` clones `choices` for internal table changes

Structures

- `history_event_entity_expels_hfst`: added (new in 0.44.11)
- `history_event_site_surrenderedst`: added (new in 0.44.11)
- `history_event_type`: added `SITE_SURRENDERED`, `ENTITY_EXPELS_HF` (new in 0.44.11)
- `syndrome`: identified a few fields
- `viewscreen_civliststst`: fixed layout and identified many fields

7.12.58 DFHack 0.44.11-alpha1

Structures

- Added support for automatically sizing arrays indexed with an enum
- Dropped 0.44.10 support
- Removed stale generated CSV files and DT layouts from pre-0.43.05
- `announcement_type`: new in 0.44.11: `NEW_HOLDING`, `NEW_MARKET_LINK`
- `breath_attack_type`: added `OTHER`
- `historical_figure_info.relationships.list`: added `unk_3a`-`unk_3c` fields at end
- `interface_key`: added bindings new in 0.44.11
- `occupation_type`: new in 0.44.11: `MESSENGER`
- `profession`: new in 0.44.11: `MESSENGER`
- **`ui_sidebar_menus`:**
  - `unit.in_squad`: renamed to `unit.squad_list_opened`, fixed location
  - `unit`: added `expel_error` and other unknown fields new in 0.44.11
  - `hospital`: added, new in 0.44.11
  - `num_speech_tokens`, `unk_17d8`: moved out of `command_line` to fix layout on x64
- `viewscreen_civliststst`: added a few new fields (incomplete)
- `viewscreen_locationsst`: identified `edit_input`

7.12.59 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)

7.12.60 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)

7.12.61 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

7.12.62 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)

7.12.63 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

7.12.64 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

7.12.65 DFHack 0.44.08-alpha1

Fixes

- *fix/dead-units*: fixed a bug that could remove some arriving (not dead) units

7.12.66 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

7.12.67 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`

7.12.68 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

7.12.69 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`

7.12.70 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

7.12.71 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

7.12.72 DFHack 0.44.03-beta1

Fixes

- *autolabor*, *autohauler*, *labormanager*: added support for "put item on display" jobs and building/destroying display furniture
- *gui/gm-editor*: fixed an error when editing primitives in Lua tables

Misc Improvements

- *devel/dump-offsets*: now ignores `index` globals
- *gui/pathable*: added tile types to sidebar
- ***modtools/skill-change*:**
  - now updates skill levels appropriately
  - only prints output if `-loud` is passed

Structures

- Added `job_type.PutItemOnDisplay`
- Added `twbt_render_map` code offset on x64
- Fixed an issue preventing `enabler` from being allocated by DFHack
- Found `renderer` vtable on osx64
- **New globals:**
  - `version`
  - `min_load_version`
  - `movie_version`

    - basic_seed
    - title
    - title_spaced
    - ui_building_resize_radius
- adventure_movement_optionst, adventure_movement_hold_tilest, adventure_movement_climbst: named coordinate fields
- mission: added type
- unit: added 3 new vmethods: getCreatureTile, getCorpseTile, getGlowTile
- viewscreen_assign_display_itemst: fixed layout on x64 and identified many fields
- viewscreen_reportliststst: fixed layout, added mission_id vector
- world.status: named missions vector

7.12.73 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

7.12.74 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

7.12.75 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

EIGHT

ABOUT DFHACK

These pages contain information about the general DFHack project.

8.1 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 Discord or the forums, or make a pull request!

Name	Github	Other
8Z	8Z	
Abel	abstern	
acwatkins	acwatkins	
Aleksandr Glotov	glotov4	
Alex Blamey	Cubittus	
Alexander Collins	gearsix	
Alexander Gavrilov	angavrilov	ag
Amber Brown	hawkowl	
Amostubal	Amostubal	
Andrea Cattaneo	acattaneo88	
AndreasPK	AndreasPK	
Andriel Chaoti	AndrielChaoti	
Angus Mezick	amezick	
Antalia	tamarakorr	
Anuradha Dissanayake	falconne	
Ariphaos	Ariphaos	
arzyu	arzyu	
Atkana	Atkana	
AtomicChicken	AtomicChicken	
Batt Mush	hobotron-df	
Bearskie	Bearskie	
belal	jimhester	
Ben Lubar	BenLubar	
Ben Rosser	TC01	
Benjamin McKenna	britishben	
Benjamin Seiller	bseiller	RedDwarfStepper
billw2012	billw2012	
BrickViking	brickviking	
brndd	brndd	burneddi

continues on next page

Table 1 – continued from previous page

Name	Github	Other
Caldfir	caldfir	
Cameron Ewell	Ozzatron	
Carter Bray	Qartar	
Chris Dombroski	cdombroski	
Chris Parsons	chrismdp	
cjhammel	cjhammel	
Clayton Hughes		
Clément Vuchener	cvuchener	
Corey	CoreyJ87	
daedsidog	daedsidog	
Dan Amlund	danamlund	
Daniel Brooks	db48x	
David	Nilsolm	
David Corbett	dscorbett	
David Seguin	dseguin	
David Timm	dtimm	
Dean Golden	LightHardt	
Deon		
dhthwy	dhthwy	
dikbut	Tjudge1	
Dmitrii Kurkin	Kurkin	
DoctorVanGogh	DoctorVanGogh	
Donald Ruegsegger	hashaash	
doomchild	doomchild	
Droseran	Droseran	
DwarvenM	DwarvenM	
Eamon Bode	eamondo2	Baron Von Munchhausen
EarthPulseAcademy	EarthPulseAcademy	
ElMendukol	ElMendukol	
ElsaTheHobo	ElsaTheHobo	Elsa
enjia2000		
Eric Wald	eswald	
Erik Youngren	Artanis	
Espen Wiborg		
expwnent	expwnent	
Feng		
figment	figment	
Gabe Rau	gaberau	
Gaelmare	Gaelmare	
gchristopher	gchristopher	
George Murray	GitOnUp	
grubsteak	grubsteak	
Guilherme Abraham	GuilhermeAbraham	
Harlan Playford	playfordh	
Hayati Ayguen	hayguen	
Herwig Hochleitner	bendlas	
Hevlikn	Hevlikn	
Ian S	kremlin-	
IndigoFenix		
Jacek Konieczny	Jajcus	

continues on next page

Table 1 – continued from previous page

Name	Github	Other
James	20k	
James Gilles	kazimuth	
James Logsdon	jlogsdon	
Janeene Beeforth	dawnmist	
Jared Adams		
Jeremy Apthorp	nornagon	
Jim Lisi	stonetoad	
Jimbo Whales	jimbowhales	
jimcarreer	jimcarreer	
jj	jjyg	jj``
Joel Meador	janxious	
John Beisley	huin	
John Cosker	johncosker	
John Shade	gsvslto	
Jonas Ask		
Jonathan Clark	AridTag	
Josh Cooper	cppcooper	coope
jowario	jowario	
kane-t	kane-t	
Kelly Kinkade	ab9rf	
Kelvie Wong	kelvie	
Kib Arekatír	arekatir	
KlonZK	KlonZK	
Kris Parker	kaypy	
Kristjan Moore	kristjanmoore	
Kromtec	Kromtec	
Kurik Amudnil		
Kévin Boissonneault	KABoissonneault	
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	
Mikhail Panov	Halifay	
Mikko Juola	Noeda	Adeon
Milo Christiansen	milochristiansen	
MithrilTuxedo	MithrilTuxedo	

continues on next page

Table 1 – continued from previous page

Name	Github	Other
mizipzor	mizipzor	
moversti	moversti	
mrrho	mrrho	
Murad Beybalaev	Erquint	
Myk Taylor	myk002	
Najeeb Al-Shabibi	master-spike	
napagokc	napagokc	
Neil Little	nmlittle	
Nick Rart	nickrart	comestible
Nicolas Ayala	nicolasayala	
Nik Nyby	nikolas	
Nikolay Amiantov	abbradar	
nocico	nocico	
NotRexButCaesar	NotRexButCaesar	
Nuno Fernandes	UnknowableCoder	
nuvu	vallode	
Omniclasm		
oorzkws	oorzkws	
OwnageIsMagic	OwnageIsMagic	
palenerd	dlmarquis	
PassionateAngler	PassionateAngler	
Patrik Lundell	PatrikLundell	
Paul Fenwick	pjf	
PeridexisErrant	PeridexisErrant	
Peter Hansen	previsualconsent	
Petr Mrázek	peterix	
Pfhreak	Pfhreak	
Pierre Lulé	plule	
Pierre-David Bélanger	pierredavidbelanger	
PopnROFL	PopnROFL	
potato		
ppaawwll	ppaawwll	
Priit Laes	plaes	
Putnam	Putnam3145	
quarque2	quarque2	
Quietust	quietust	_Q
Rafał Karczmarczyk	CarabusX	
Raidau	Raidau	
Ralph Bisschops	ralpha	
Ramblurr	Ramblurr	
rampaging-poet		
Raoul van Putten		
Raoul XQ	raoulxq	
reverb		
Rich Rauenzahn	rrauenza	
Rinin	Rinin	
rndmvar	rndmvar	
Rob Bailey	actionninja	
Rob Goodberry	robob27	
Robert Heinrich	rh73	

continues on next page

Table 1 – continued from previous page

Name	Github	Other
Robert Janetzko	robertjanetzko	
Rocco Moretti	roccomoretti	
RocheLimit		
rofl0r	rofl0r	
root		
Rose	RosaryMala	
Roses	Pheosics	
Ross M	RossM	
rout		
Roxy	TealSeer	gallowsCalibrator
rubybrowncoat	rubybrowncoat	
Rumrusher	rumrusher	
RusAnon	RusAnon	
Ryan Bennitt	ryanbennitt	
Ryan Dwyer	ToxicBananaParty	Jimdude2435
Ryan Williams	Bumber64	Bumber
sami		
scamtank	scamtank	
Scott Ellis	StormCrow42	
Sebastian Wolfertz	Enkrod	
SeerSkye	SeerSkye	
seishuuu	seishuuu	
Seth Woodworth	sethwoodworth	
shevernitskiy	shevernitskiy	
Shim Panze	Shim-Panze	
Silver	silverflyone	
simon		
Simon Jackson	sizeak	
Simon Lees	simotek	
stolencatkarma		
Stoyan Gaydarov	sgayda2	
Su	Moth-Tolias	
suokko	suokko	shrieker
sv-esk	sv-esk	
Tachytaenius	wolfboyft	
Tacomagic		
TaxiService	TaxiService	
terribleperson	terribleperson	
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	
Tom Prince		

continues on next page

Table 1 – continued from previous page

Name	Github	Other
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 H	TSM-EVO	
Will Rogers	wjrogers	
WoosterUK	WoosterUK	
XianMaeve	XianMaeve	
ZechyW	ZechyW	
Zhentar	Zhentar	
zilpin	zilpin	
Zishi Wu	zishiwu123	

8.2 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.

Some graphic assets are derived from vanilla DF assets and used with permission from Bay12.

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.
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
googletest	BSD 3-Clause	(c) 2008, Google Inc.

8.2.1 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.
```

8.2.2 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.
```

8.2.3 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
```

8.3 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

- *fix/fat-dwarves*
- *fix/feeding-timers*
- *fix/merchants*
- *fortplan*
- *gui/assign-rack*
- *gui/automelt*
- *gui/create-tree*
- *gui/dig*
- *gui/hack-wish*
- *gui/manager-quantity*
- *gui/mechanisms*
- *gui/no-dfhack-init*
- *masspit*
- *mousequery*
- *petcapRemover*
- *resume*
- *ruby*
- *search*
- *show-unit-syndromes*
- *stocksettings*
- *title-version*
- *unsuspend*
- *warn-starving*
- *warn-stealers*
- *warn-stuck-trees*
- *workorder-recheck*

8.3.1 autohauler

An automated labor management tool that only addressed hauling labors, leaving the assignment of skilled labors entirely up to the player. Fundamentally incompatible with the work detail system of labor management in v50 of Dwarf Fortress.

8.3.2 automaterial

Moved frequently used materials to the top of the materials list when building buildings. Also offered extended options when building constructions. All functionality has been merged into *buildingplan*.

8.3.3 automelt

Automatically mark items for melting when they are brought to a monitored stockpile. Merged into *logistics*.

8.3.4 autotrade

Automatically mark items for trading when they are brought to a monitored stockpile. Merged into *logistics*.

8.3.5 autounsuspend

Replaced by *suspendmanager*.

8.3.6 combine-drinks

Replaced by the new *combine* script. Run `combine here --types=drink`

8.3.7 combine-plants

Replaced by the new *combine* script. Run `combine here --types=plants`

8.3.8 command-prompt

Replaced by *gui/launcher –minimal*.

8.3.9 create-items

Replaced by *gui/create-item*.

8.3.10 deterioriateclothes

Replaced by the new combined *deteriorate* script. Run `deteriorate --types=clothes`.

8.3.11 deterioratecorpses

Replaced by the new combined *deteriorate* script. Run `deteriorate --types=corpses`.

8.3.12 deterioratefood

Replaced by the new combined *deteriorate* script. Run `deteriorate --types=food`.

8.3.13 devel/unforbidall

Replaced by the *unforbid* script. Run `unforbid all --quiet` to match the behavior of the original `devel/unforbidall` script.

8.3.14 digfort

A script to designate an area for digging according to a plan in csv format. Please use DFHack's more powerful *quickfort* script instead. You can use your existing .csv files. Just move them to the `blueprints` folder in your DF installation, and instead of `digfort file.csv`, run `quickfort run file.csv`.

8.3.15 drain-aquifer

Replaced by *aquifer* and *gui/aquifer*.

8.3.16 embark-tools

Replaced by *gui/embark-anywhere*. Other functionality was replaced by the DF v50 UI.

8.3.17 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.

8.3.18 fix/build-location

The corresponding DF Bug 5991 was fixed in DF 0.40.05.

8.3.19 fix/diplomats

The corresponding DF Bug 3295 was fixed in DF 0.40.05.

8.3.20 fix/fat-dwarves

The corresponding DF Bug 5971 was fixed in DF 0.40.05.

8.3.21 fix/feeding-timers

The corresponding DF Bug 2606 was fixed in DF 0.40.12.

8.3.22 fix/merchants

Humans can now make trade agreements. This fix is no longer necessary.

8.3.23 fortplan

Designates furniture for building according to a `.csv` file with quickfort-style syntax. Please use DFHack's more powerful *quickfort* script instead. You can use your existing .csv files. Just move them to the `blueprints` folder in your DF installation, and instead of `fortplan file.csv` run `quickfort run file.csv`.

8.3.24 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.

8.3.25 gui/automelt

Replaced by the *stockpiles* overlay and the gui for *logistics*.

8.3.26 gui/create-tree

Replaced by *gui/sandbox*.

8.3.27 gui/dig

Renamed to gui/design

8.3.28 gui/hack-wish

Replaced by *gui/create-item*.

8.3.29 gui/manager-quantity

Ability to modify manager order quantities has been added to the vanilla UI.

8.3.30 gui/mechanisms

Linked building interface has been added to the vanilla UI.

8.3.31 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.

8.3.32 masspit

Replaced with a GUI version: *gui/masspit*.

8.3.33 mousequery

Functionality superseded by vanilla v50 interface.

8.3.34 petcapRemover

Renamed to *pet-uncapper*.

8.3.35 resume

Allowed you to resume suspended jobs and displayed an overlay indicating suspended building construction jobs. Replaced by *unsuspend* script.

8.3.36 ruby

Support for the Ruby language in DFHack scripts was removed due to the issues the Ruby library causes when used as an embedded language.

8.3.37 search

Functionality was merged into *sort*.

8.3.38 show-unit-syndromes

Replaced with a GUI version: *gui/unit-syndromes*.

8.3.39 stocksettings

Along with `copystock`, `loadstock` and `savestock`, replaced with the new *stockpiles* API.

8.3.40 title-version

Replaced with an *overlay*.

8.3.41 unsuspend

Merged into *suspendmanager*.

8.3.42 warn-starving

Functionality was merged into *gui/notify*.

8.3.43 warn-stealers

Functionality was merged into *gui/notify*.

8.3.44 warn-stuck-trees

The corresponding DF Bug 9252 was fixed in DF 0.44.01.

8.3.45 workorder-recheck

Tool to set 'Checking' status of the selected work order, allowing conditions to be reevaluated. Merged into *orders*.

8.4 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

8.4.1 DFHack 50.13-r2

New Tools

- **Updated for adventure mode:**
  - *reveal*
  - *gui/sandbox*, *gui/create-item*, *gui/reveal*
- *adaptation*: (reinstated) inspect or set unit cave adaptation levels
- *fix/engravings*: fix corrupt engraving tiles
- *flashstep*: (reinstated) teleport your adventurer to the mouse cursor
- *ghostly*: (reinstated) allow your adventurer to phase through walls
- *markdown*: (reinstated) export description of selected unit or item to a text file
- *resurrect-adv*: (reinstated) allow your adventurer to recover from death
- *reveal-adv-map*: (reinstated) reveal (or hide) the adventure map
- *unretire-anyone*: (reinstated) choose anybody in the world as an adventurer

New Features

- DFHack and the Dwarf Fortress translation project can now both be run at the same time
- *buildingplan*: quick material filter favorites on main planner panel
- *instruments*: new subcommand `instruments order` for creating instrument work orders

Fixes

- *blueprint*: correctly define stockpile boundaries in recorded stockpile ("place") blueprints when there are adjacent non-rectangular stockpiles of identical types
- *caravan*: don't include undiscovered divine artifacts in the goods list
- *combine*: respect container volume limits
- ***dig*:**
  - refresh count of tiles that will be modified by "mark all designated tiles on this z-level for warm/damp dig" when the z-level changes
  - don't affect already-revealed tiles when marking z-level for warm/damp dig
- *gui/quantum*: fix processing when creating a quantum dump instead of a quantum stockpile
- *logistics*: include semi-wild pets when autoretrain is enabled
- *modtools/create-item*: now functions properly when the `reaction-gloves` tweak is active
- *prospector*: don't use scientific notation for representing large numbers
- ***quickfort*:**
  - don't designate multiple tiles of the same tree for chopping when applying a tree chopping blueprint to a multi-tile tree
  - fix detection of valid tiles for wells
- *suspendmanager*: fully suspend unbuildable dead ends (e.g. buildling second level of a wall when the wall top is only accessible via ramp, causing the planned wall to be pathable but not buildable)
- ***zone*:**
  - fix display of distance from cage/pit for small pets in assignment dialog
  - refresh values in distance column when switching selected pastures when the assign animals dialog is open

Misc Improvements

- Dreamfort: move wells on services level so brawling drunken tavern patrons are less likely to fall in
- New commandline options for controlling the Cloud Save coprocess when launching from Steam. See the *DFHack Core* documentation for details.
- *caravan*: display who is in the cages you are selecting for trade and whether they are hostile
- *combine*: reduce combined drink sizes to 25
- *deathcause*: automatically find and choose a corpse when a pile of mixed items is selected
- ***dig*:**
  - warm/damp/aquifer status will now be shown in mining mode for tiles that your dwarves can see from the level below
  - warm/damp/aquifer status will now be shown when in smoothing/engraving modes
- *flashstep*: new keybinding for teleporting adventurer to the mouse cursor: Ctrl-t (when adventure map is in the default state and mortal mode is disabled in DFHack preferences)
- *gui/autobutcher*: add shortcuts for butchering/unbutchering all animals

- *gui/launcher*: add button for copying output to the system clipboard
- ***gui/quantum*:**
  - add option for whether a minecart automatically gets ordered and/or attached
  - when attaching a minecart, show which minecart was attached
  - allow multiple feeder stockpiles to be linked to the minecart route
- *markdown*: new keybinding for triggering text export: Ctrl-t (when unit or item is selected)
- *prioritize*: add PutItemOnDisplay jobs to the default prioritization list – when these kinds of jobs are requested by the player, they generally want them done ASAP
- ***regrass*:**
  - can now add grass to stairs, ramps, ashes, buildings, muddy stone, shrubs, and trees
  - can now restrict area of effect to specified tile, block, cuboid, or z-levels
  - can now add grass in map blocks where there hasn't been any
  - can now choose specific grass type
- *stockpiles*: support import and export "desired items" configuration for route stops
- *unretire-anyone*: new keybinding for adding a historical figure to the adventurer selection list in the adventure mode setup screen: Ctrl-a

API

- `dfhack.items.getReadableDescription()`: easy API for getting a human-readable item description with useful annotations and information (like tattered markers or who is in a cage)
- `Items::createItem`: now returns a list of item pointers rather than a single ID, moved creator parameter to beginning, added growth_print and no_floor parameters at end
- `World::getAdventurer`: returns current adventurer unit
- `World::ReadPauseState`: now returns true when the game is effectively paused due to a large panel obscuring the map. this aligns the return value with the visual state of the pause button when in fort mode.

Lua

- `dfhack.internal.setClipboardTextCp437Multiline`: for copying multiline text to the system clipboard
- `dfhack.items.createItem`: return value and parameters have changed as per C++ API
- `dfhack.world.getAdventurer`: returns current adventurer unit

Documentation

- Quickfort Blueprint Library: add demo videos for pump stack and light aquifer tap blueprints
- Update docs for dependency requirements and compilation procedures

8.4.2 DFHack 50.13-r1.1

Fixes

- *deathcause*: fix error on run
- *gui/quantum*: accept all item types in the output stockpile as intended

Documentation

- Update docs on release procedures and symbol generation

8.4.3 DFHack 50.13-r1

New Tools

- *gui/quantum*: (reinstated) point and click interface for creating quantum stockpiles or quantum dumps
- *gui/unit-info-viewer*: (reinstated) give detailed information on a unit, such as egg laying behavior, body size, birth date, age, and information about their afterlife behavior (if a ghost)

Fixes

- Fixed incorrect DFHack background window texture when DF is started in ascii mode and subsequently switched to graphics mode
- Fixed misidentification of visitors from your own civ as residents; affects all tools that iterate through citizens/residents
- *cursecheck*: act on selected unit only if a unit is selected
- *exterminate*: don't classify dangerous non-invader units as friendly (e.g. snatchers)
- ***gui/create-item*:**
  - properly restrict bags to bag materials by default
  - allow gloves and shoees to be made out of textiles by default
- *open-legends*: don't interfere with the dragging of vanilla list scrollbars

Misc Improvements

- *gui/gm-unit*: changes to unit appearance will now immediately be reflected in the unit portrait
- *open-legends*: allow player to cancel the "DF will now exit" dialog and continue browsing
- *suspendmanager*: Account for walls planned on the z-layer below when determining accessibility to a job

Structures

- `biome_type`: add enum attrs for `caption` and `plant_raw_flags`

Documentation

- *autoclothing*: add section comparing `autoclothing` and *tailor* to guide players choosing which to enable

8.4.4 DFHack 50.12-r3

New Tools

- *aquifer*: commandline tool for creating, draining, and modifying aquifers
- *gui/aquifer*: interactive aquifer visualization and editing
- *open-legends*: (reinstated) open legends mode directly from a loaded fort

New Features

- ***blueprint*:**
  - designations and active dig jobs are now captured in generated blueprints
  - warm/damp dig markers are captured in generated blueprints
- *buildingplan*: add overlays for unlinking and freeing mechanisms from buildings
- ***dig*:**
  - designate tiles for damp or warm dig, which allows you to dig through damp or warm tiles without designations being canceled
  - damp and warm tile icons now remain visible when included in the designation selection box (graphics mode)
  - aquifer tiles are now visually distinct from "just damp" tiles (graphics and ascii modes)
  - light aquifer tiles are now visually distinct from heavy aquifer tiles (graphics and ascii modes)
  - autodig designations that are marked for damp/warm dig propagate the damp/warm tag when expanding to newly exposed tiles
- *gui/notify*: optional notification for general wildlife (not on by default)
- *gui/quickfort*: add options for setting warm/damp dig markers when applying blueprints
- *gui/reveal*: new "aquifer only" mode to only see hidden aquifers but not reveal any tiles
- *quickfort*: add options for setting warm/damp dig markers when applying blueprints

Fixes

- fix behavior of Linux Steam launcher on systems that don't support the inotify API
- fix rendering of resize "notch" in lower right corner of resizable windows in ascii mode
- *agitation-rebalance*: fix calculated percent chance of cavern invasion
- *armoks-blessing*: fix error when making "Normal" attributes legendary
- *emigration*: remove units from burrows when they emigrate
- *fix/loyaltycascade*: fix edge case where loyalties of renegade units were not being fixed
- *gui/launcher*: don't pop up a result dialog if a command run from minimal mode has no output
- ***quickfort*:**
  - stockpiles can now be placed even if there is water covering the tile, as per vanilla behavior
  - reject tiles for building that contain magma or deep water
- *stonesense*: fix a crash with buildings made of unusual materials (such as campsite tents made out of organic "walls")
- *suspendmanager*: prevent cancellation spam when an item is preventing a building from being completed

Misc Improvements

- aquifer_tap blueprint: now designates in damp dig mode for uninterrupted digging in a light aquifer
- pump_stack blueprint: now designates in warm and damp dig mode for uninterrupted digging through warm and damp tiles
- *agitation-rebalance*: when more than the maximum allowed cavern invaders are trying to enter the map, prefer keeping the animal people invaders instead of their war animals
- *gui/control-panel*: add alternate "nodump" version for *cleanowned* that does not cause citizens to toss their old clothes in the dump. this is useful for players who would rather sell old clothes than incinerate them
- *gui/reveal*: show aquifers even when not in mining mode
- *keybinding*: you can now assign keybindings to mouse buttons (if your mouse has more than the three buttons already used by DF)
- *tailor*: allow turning off automatic confiscation of tattered clothing

Removed

- *drain-aquifer*: replaced by `aquifer drain --all`; an alias now exists so `drain-aquifer` will automatically run the new command

API

- `Buildings::checkFreeTiles`: now takes a `allow_flow` parameter to control whether water- or magma-filled tiles are valid
- `Units::citizensRange`: c++-20 std::range filter for citizen units
- `Units::forCitizens`: iterator callback function for citizen units
- `Units::paintTile`, `Units::readTile`: now takes an optional field specification for reading and writing to specific map compositing layers

Lua

- `dfhack.gui.matchFocusString`: focus string matching is now case sensitive (for performance reasons)

Structures

- name many previously-unknown map-related fields and flag bits
- `job_type`: new job class type: "Carving" (for smoothing and detailing)
- `unit_action_data_attack` (`unit_move_attackst`): identify flags

Documentation

- Lua API: documented existing `enum:next_item(index)` function

8.4.5 DFHack 50.12-r2.1

Fixes

- *control-panel*: properly auto-enable newly added bugfixes
- *fix/noexert-exhaustion*: fix typo in control panel registry entry which prevented the fix from being run when enabled
- *gui/suspendmanager*: fix script startup errors
- *orders*: don't intercept keyboard input for setting skill or labor restrictions on workshop workers tab when the player is setting the building nickname

Misc Improvements

- *gui/unit-syndromes*: make syndromes searchable by their display names (e.g. "necromancer")

8.4.6 DFHack 50.12-r2

New Tools

- *agitation-rebalance*: alter mechanics of irriation-related attacks so they are less constant and are more responsive to recent player bahavior
- *devel/block-borders*: (reinstated) highlights boundaries of map blocks or embark tile blocks
- *fix/noexert-exhaustion*: fix "Tired" NOEXERT units. Enabling via *gui/control-panel* prevents NOEXERT units from getting stuck in a "Tired" state
- *fix/ownership*: fix instances of multiple citizens claiming the same items, resulting in "Store owned item" job loops
- *fix/stuck-worship*: fix prayer so units don't get stuck in uninterruptible "Worship!" states
- *instruments*: provides information on how to craft the instruments used by the player civilization
- *modtools/if-entity*: (reinstated) modder's resource for triggering scripted content depending on the race of the loaded fort
- *modtools/item-trigger*: (reinstated) modder's resource for triggering scripted content when specific items are used

New Features

- *exterminate*: new "disintegrate" kill method that additionally destroys carried items
- *gui/settings-manager*: add import, export, and autoload for work details
- *logistics*: `autoretrain` will automatically assign trainers to your partially-trained (but not yet domesticated) livestock. this prevents children of partially-trained parents from reverting to wild if you don't notice they were born
- *orders*: add overlay for configuring labor and skill level restrictions for workshops
- *quickfort*: allow setting of workshop profile properties (e.g. labor, skill restrictions) from build blueprints
- *sort*: updated and reinstated military status/squad membership/burrow membership filter for work animal assignment screen
- *stocks*: add button/hotkey for removing empty categories from the stocks list

Fixes

- *autochop*: fix underestimation of log yield for cavern mushrooms
- *autoclothing*: don't produce clothes for dead units
- *caravan*: fix trade price calculations when the same item was requested for both import and export
- *catsplosion*: only cause pregnancies in adults
- *control-panel*: fix filtering not filtering when running the `list` command
- ***gui/launcher*:**
  - fix detection on Shift-Enter for running commands and autoclosing the launcher
  - fix history scanning (Up/Down arrow keys) being slow to respond when in minimal mode
- ***gui/notify*:**

  - prevent notification overlay from showing up in arena mode
  - don't zoom to forbidden depots for merchants ready to trade notification
- ***logistics*:**
  - don't melt/trade/dump empty containers that happen to be sitting on the stockpile unless the stockpile accepts those item types
  - don't send autotrade items to forbidden depots

Misc Improvements

- **Dreamfort:**
  - the four Craftsdwarf's workshops on the industry level are now specialized for Stonecrafting, Woodcrafting, Bone Carving, and miscellaneous tasks, respectively
  - update embark profile recommendations and example embark profile
- Many tools that previously only worked for citizens or only for dwarves now work for all citizens and residents, e.g. *fastdwarf*, *rejuvenate*, etc.
- When launched from the Steam client on Linux, both Dwarf Fortress and DFHack will be shown as "Running". This ensures that DF has proper accounting for Linux player usage.
- ***allneeds*:**
  - select a dwarf in the UI to see a summary of needs for just that dwarf
  - provide options for sorting the cumulative needs by different criteria
- *autobutcher*: prefer butchering partially trained animals and save fully domesticated animals to assist in wildlife domestication programs
- *autodump*: can now teleport items loosely stored in buildings (clutter)
- ***buildingplan*:**
  - remember player preference for whether unavailable materials should be hidden in the filter selection dialog
  - sort by available quantity by default int he filter selection dialog
- *cleaners*: protect farm plots when cleaning mud
- *control-panel*: enable tweaks quietly on fort load so we don't spam the console
- *devel/tile-browser*: simplify interface now that SDL automatically normalizes texture scale
- ***dwarfvet*:**
  - automatically unassign animals from pastures when they need treatment so they can make their way to the hospital. reassign them to their original pasture when treatment is complete.
  - ignore animals assigned to cages or restraints
- *exterminate*: make race name matching case and space insensitive
- *gui/gm-editor*: support opening engraved art for inspection
- ***gui/launcher*:**
  - add interface for browsing and filtering commands by tags
  - add support for history search (Alt-s hotkey) when in minimal mode

    - add support for the `clear` command and clearing the scrollback buffer
- *gui/notify*: Shift click or Shift Enter on a zoomable notification to zoom to previous target
- *gui/teleport*: add global Ctrl-Shift-T keybinding (only avaiable when DFHack mortal mode is disabled)
- *prioritize*: print out custom reaction and hauling jobs in the same format that is used for `prioritize` command arguments so the player can just copy and paste
- *suspendmanager*: improve performance when there are many active jobs
- *tweak*: add `quiet` option for silent enablement and disablement of tweaks

API

- `Units::getCitizens`: now includes residents by default
- `Units::isForgottenBeast`: property check for forgotten beasts
- `Units::isGreatDanger`: now includes forgotten beasts
- `Units::isResident`: property check for residents (as opposed to citizens)

Lua

- *helpdb*: `search_entries` now returns a match if *all* filters in the `include` list are matched. previous behavior was to match if *any* `include` filter matched.
- `dfhack.units.getCitizens`: now includes residents by default
- `dfhack.units.isForgottenBeast`: make new units method available to Lua
- `matinfo.decode`: now directly handles plant objects
- `widgets.Label`: `*pen` attributes can now either be a pen or a function that dynamically returns a pen

Structures

- `activity_event`: identify fields and type values
- `plant_tree_info`: define tree body and branch flags
- `plotinfo.hauling`: name fields related to the hauling route panel
- `unit`: identify and define many previously unknown fields, types, and enums

Documentation

- *Introduction and overview*: refresh getting started content
- *DFHack overlay dev guide*: updated examples and troubleshooting steps
- *Quickstart guide*: refresh quickstart guide

8.4.7 DFHack 50.12-r1.1

Fixes

- *sort*: fix crash when assigning work animals to units

Removed

- offline HTML rendered docs are no longer distributed with DFHack since they are randomly triggering Windows Defender antivirus heuristics. If you want to download DFHack docs for offline browsing, you can still get them from the Downloads link at https://dfhack.org/docs

8.4.8 DFHack 50.12-r1

Fixes

- *gui/design*: no longer comes up when Ctrl-D is pressed but other DFHack windows have focus
- *gui/notify*: persist notification settings when toggled in the UI

Misc Improvements

- *gui/launcher*: developer mode hotkey restored to Ctrl-D
- *sort*: squad assignment overlay rewritten for compatiblity with new vanilla data structures and screen layouts

Removed

- *burrow*: removed overlay 3D box select since it is now provided by the vanilla UI
- *sort*: removed Search widgets for screens that now have vanilla search

API

- `Gui::getWidget`: retrieve a vanilla DF widget by name or index

Lua

- `dfhack.gui.getWidgetChildren`: retrieve a list of child widgets for a given widget container
- `dfhack.gui.getWidget`: retrieve a vanilla DF widget by hierarchy path, with each step specified by a widget name or index

8.4.9 DFHack 50.11-r7

New Tools

- *add-thought*: (reinstated) add custom thoughts to a dwarf
- *combat-harden*: (reinstated) set a dwarf's resistence to being affected by visible corpses
- *devel/input-monitor*: interactive UI for debugging input issues
- *gui/notify*: display important notifications that vanilla doesn't support yet and provide quick zoom links to notification targets.
- *gui/petitions*: (reinstated) show outstanding (or all historical) petition agreements for guildhalls and temples
- *list-waves*: (reinstated) show migration wave information
- *make-legendary*: (reinstated) make a dwarf legendary in specified skills
- *pet-uncapper*: (reinstated, renamed from `petcapRemover`) allow pets to breed beyond the default population cap of 50
- *tweak*: (reinstated) a collection of small bugfixes and gameplay tweaks
- *undump-buildings*: (reinstated) remove dump designation from in-use building materials

New Features

- *cleanowned*: Add a "nodump" option to allow for confiscating items without dumping
- *tweak*: Add "flask-contents", makes flasks/vials/waterskins be named according to their contents

Fixes

- *autoclothing*: Fix enabled behavior
- *caravan*: display book and scroll titles in the goods and trade dialogs instead of generic scroll descriptions
- *dig-now*: fix digging stairs in the surface sometimes creating underworld gates.
- *dig*: overlay that shows damp designations in ASCII mode now propertly highlights tiles that are damp because of an aquifer in the layer above
- *fix/retrieve-units*: prevent pulling in duplicate units from offscreen
- *gui/blueprint*: changed hotkey for setting blueprint origin tile so it doesn't conflict with default map movement keys
- *gui/control-panel*: fix error when toggling autostart settings
- *gui/design*: clicking the center point when there is a design mark behind it will no longer simultaneously enter both mark dragging and center dragging modes. Now you can click once to move the shape, and click twice to move only the mark behind the center point.
- *item*: avoid error when scanning items that have no quality rating (like bars and other construction materials)
- *source*: fix issue where removing sources would make some other sources inactive
- *strangemood*: correctly recognize Stonecutter and Stone Carver as moodable skills, move the Mason's boosted mood chance to the Stone Carver, and select Fell/Macabre based on long-term stress
- ***warn-stranded*:**

  - don't complain about units that aren't on the map (e.g. soldiers out on raids)
  - when there was at least one truly stuck unit and miners were actively mining, the miners were also confusingly shown in the stuck units list
- `gui.View:getMouseFramePos`: function now detects the correct coordinates even when the widget is nested within other frames
- `Gui::makeAnnouncement`, `Gui::autoDFAnnouncement`: don't display popup for all announcement types
- `Gui::revealInDwarfmodeMap`: properly center the zoom even when the target tile is near the edge of the map
- `Units::getVisibleName`: don't reveal the true identities of units that are impersonating other historical figures

Misc Improvements

- *autonestbox*: assign egg layers to the nestbox they have chosen if they have already chosen a nestbox
- *buildingplan*: use closest matching item rather than newest matching item
- *caravan*: move goods to trade depot dialog now allocates more space for the display of the value of very expensive items
- *exportlegends*: make progress increase smoothly over the entire export and increase precision of progress percentage
- *extinguish*: allow selecting units/items/buildings in the UI to target them for extinguishing; keyboard cursor is only required for extinguishing map tiles that cannot be selected any other way
- *gui/autobutcher*: ask for confirmation before zeroing out targets for all races
- *gui/mod-manager*: will automatically unmark the default mod profile from being the default if it fails to load (due to missing or incompatible mods)
- ***gui/quickfort*:**
  - can now dynamically adjust the dig priority of tiles designated by dig blueprints
  - can now opt to apply dig blueprints in marker mode
- ***item*:**
  - change syntax so descriptions can be searched for without indicating the `--description` option. e.g. it's now `item count royal` instead of `item count --description royal`
  - add `--verbose` option to print each item as it is matched
- *probe*: act on the selected building/unit instead of requiring placement of the keyboard cursor for `bprobe` and `cprobe`
- *regrass*: also regrow depleted cavern moss
- ***zone*:**
  - animal assignment dialog now shows distance to pasture/cage and allows sorting by distance
  - animal assignment dialog shows number of creatures assigned to this pasture/cage/etc.

Removed

- *gui/create-tree*: replaced by *gui/sandbox*
- *gui/manager-quantity*: the vanilla UI can now modify manager order quantities after creation
- *warn-starving*: combined into *gui/notify*
- *warn-stealers*: combined into *gui/notify*

API

- Gui focus strings will now include `dwarfmode/Default` if the only other panel open is the Squads panel
- Gui module Announcement functions now use DF's new announcement alert system
- `Gui::addCombatReport`, `Gui::addCombatReportAuto`: add versions that take `report *` instead of report vector index
- `Gui::MTB_clean`, `Gui::MTB_parse`, `Gui::MTB_set_width`: new functions for manipulating `markup_text_boxst`
- `Gui::revealInDwarfmodeMap`: unfollow any currently followed units/items so the viewport doesn't just jump back to where it was
- `toupper_cp437(char)`, `tolower_cp437(char)`: new `MiscUtils` functions, return a char with case changed, respecting CP437
- `toUpper`, `toLower`: `MiscUtils` functions renamed to `toUpper_cp437` and `toLower_cp437`, CP437 compliant

Lua

- Overlay framework now respects `active` and `visible` widget attributes
- `dfhack.gui` announcement functions use default arguments when omitted
- `dfhack.units.getCitizens` now only returns units that are on the map
- `dfhack.upperCp437(string)`, `dfhack.lowerCp437(string)`: new functions, return string with all chars changed, respecting CP437 code page

Structures

- `buildings_other`: add correct types for civzone building vectors
- `job_skill`: correct `moodable` property for several professions

8.4.10 DFHack 50.11-r6

New Features

- *zone*: Add overlay for toggling butchering/gelding/adoption/taming options in animal "Overview" tabs

Fixes

- ***dig-now***:
  - remove diagonal ramps rendered unusable by digging
  - fix error propagating "light" and "outside" properties to newly exposed tiles when piercing the surface
- *item*: fix missing item categories when using `--by-type`
- *makeown*: fix error when adopting units that need a historical figure to be created
- *sort*: fix potential crash when switching between certain info tabs
- *suspendmanager*: overlays for suspended building info panels no longer disappear when another window has focus

Misc Improvements

- *autonestbox*: don't automatically assign partially trained egg-layers to nestboxes if they don't have an ongoing trainer assigned since they might revert to wild
- *buildingplan*: replace `[edit filters]` button in planner overlay with abbreviated filter information
- *reveal*: automatically reset saved map state when a new save is loaded

Removed

- `nopause`: functionality has moved to *spectate*

API

- `Gui::getAnyJob`: get the job associated with the selected game element (item, unit, workshop, etc.)
- `Gui::getAnyWorkshopJob`: get the first job associated with the selected workshop
- `Units::assignTrainer`: assign a trainer to a trainable animal
- `Units::unassignTrainer`: unassign a trainer from an animal

Lua

- `dfhack.gui.getAnyJob`: expose API to Lua
- `dfhack.gui.getAnyWorkshopJob`: expose API to Lua
- `dfhack.units.assignTrainer`: expose API to Lua
- `dfhack.units.isTamable`: return false for invaders to match vanilla logic
- `dfhack.units.unassignTrainer`: expose API to Lua

Structures

- `soundst`: fix alignment

8.4.11 DFHack 50.11-r5

New Tools

- *control-panel*: new commandline interface for control panel functions
- *gui/biomes*: visualize and inspect biome regions on the map
- ***gui/embark-anywhere*:**
  - new keybinding (active when choosing an embark site): Ctrl-A
  - bypass those pesky warnings and embark anywhere you want to
- *gui/reveal*: temporarily unhide terrain and then automatically hide it again when you're ready to unpause
- *gui/teleport*: mouse-driven interface for selecting and teleporting units
- *item*: perform bulk operations on groups of items.
- *uniform-unstick*: (reinstated) force squad members to drop items that they picked up in the wrong order so they can get everything equipped properly

New Features

- *gui/mass-remove*: new global keybinding: Ctrl-M while on the fort map
- *gui/settings-manager*: save and load embark difficulty settings and standing orders; options for auto-load on new embark
- *sort*: search and sort for the "choose unit to elevate to the barony" screen. units are sorted by the number of item preferences they have and the units are annotated with the items that they have preferences for
- *uniform-unstick*: add overlay to the squad equipment screen to show a equipment conflict report and give you a one-click button to (attempt to) fix
- *zone*: add button to location details page for retiring unused locations

Fixes

- DFHack tabs (e.g. in *gui/control-panel*) are now rendered correctly when there are certain vanilla screen elements behind them
- Dreamfort: fix holes in the "Inside+" burrow on the farming level (burrow autoexpand is interrupted by the pre-dug miasma vents to the surface)
- When passing map movement keys through to the map from DFHack tool windows, also pass fast z movements (shift-scroll by default)
- *ban-cooking*: fix banning creature alcohols resulting in error
- ***buildingplan*:**
  - when you save a game and load it again, newly planned buildings are now correctly placed in line after existing planned buildings of the same type

  - treat items in wheelbarrows as unavailable, just as vanilla DF does. Make sure the *fix/empty-wheelbarrows* fix is enabled so those items aren't permanently unavailable!
  - show correct number of materials required when laying down areas of constructions and some of those constructions are on invalid tiles
- *caravan*: ensure items are marked for trade when the move trade goods dialog is closed even when they were selected and then the list filters were changed such that the items were no longer actively shown
- *confirm*: properly detect clicks on the remove zone button even when the unit selection screen is also open (e.g. the vanilla assign animal to pasture panel)
- *empty-bin*: now correctly sends ammunition in carried quivers to the tile underneath the unit instead of teleporting them to an invalid (or possibly just far away) location
- ***fastdwarf*:**
  - prevent units from teleporting to inaccessible areas when in teledwarf mode
  - allow units to meander and satisfy needs when they have no current job and teledwarf mode is enabled
- *getplants*: fix crash when processing mod-added plants with invalid materials
- ***gui/design*:**
  - fix incorrect highlight when box selecting area in ASCII mode
  - fix incorrect dimensions being shown when you're placing a stockpile, but a start coordinate hasn't been selected yet
- *misery*: fix error when changing the misery factor
- *quickfort*: if a blueprint specifies an up/down stair, but the tile the blueprint is applied to cannot make an up stair (e.g. it has already been dug out), still designate a down stair if possible
- *reveal*: now avoids revealing blocks that contain divine treasures, encased horrors, and deep vein hollows (so the surprise triggers are not triggered prematurely)
- ***sort*:**
  - fix mouse clicks falling through the squad assignment overlay panel when clicking on the panel but not on a clickable widget
  - fix potential crash when removing jobs directly from the Tasks info screen
- *source*: water and magma sources and sinks now persist with fort across saves and loads
- *stonesense*: fix crash in cleanup code after mega screenshot (Ctrl-F5) completes; however, the mega screenshot will still make stonesense unresponsive. close and open the stonesense window to continue using it.
- *suspendmanager*: correctly handle building collisions with smoothing designations when the building is on the edge of the map
- *warn-stranded*: don't warn for citizens who are only transiently stranded, like those on stepladders gathering plants or digging themselves out of a hole
- `Maps::getBiomeType`, `Maps::getBiomeTypeWithRef`: fix identification of tropical oceans

Misc Improvements

- Dreamfort: put more chairs adjacent to each other to make the tavern more "social"
- The "PAUSE FORCED" badge will blink briefly to draw attention if the player attempts to unpause when a DFHack tool window requires the game to be paused
- wherever units are listed in DFHack tools, properties like "agitated" or (-trained-) are now shown
- *autochop*: better error output when target burrows are not specified on the commandline
- *autoclothing* : now does not consider worn (x) clothing as usable/available; reduces overproduction when using *tailor* at same time
- *buildingplan*: add option for preventing constructions from being planned on top of existing constructions (e.g. don't build floors on top of floors)
- *burrow*: flood fill now requires an explicit toggle before it is enabled to help prevent accidental flood fills
- ***confirm*:**
  - updated confirmation dialogs to use clickable widgets and draggable windows
  - added confirmation prompt for right clicking out of the trade agreement screen (so your trade agreement selections aren't lost)
  - added confirmation prompts for irreversible actions on the trade screen
  - added confirmation prompt for deleting a uniform
  - added confirmation prompt for convicting a criminal
  - added confirmation prompt for re-running the embark site finder
  - added confirmation prompt for reassigning or clearing zoom hotkeys
  - added confirmation prompt for exiting the uniform customization page without saving
- *fastdwarf* : now saves its state with the fort
- *fix/stuck-instruments*: now handles instruments that are left in the "in job" state but that don't have any actual jobs associated with them
- *gui/autobutcher*: interface redesigned to better support mouse control
- ***gui/control-panel*:**
  - reduce frequency for *warn-stranded* check to once every 2 days
  - tools are now organized by type: automation, bugfix, and gameplay
- ***gui/launcher*:**
  - now persists the most recent 32KB of command output even if you close it and bring it back up
  - make autocomplete case insensitive
- ***gui/mass-remove*:**
  - can now differentiate planned constructions, stockpiles, and regular buildings
  - can now remove zones
  - can now cancel removal for buildings and constructions
- *gui/quickcmd*: clickable buttons for command add/remove/edit operations
- *orders*: reduce prepared meal target and raise booze target in `basic` importable orders in the orders library

- ***sort*:**
  - add "Toggle all filters" hotkey button to the squad assignment panel
  - rename "Weak mental fortitude" filter to "Dislikes combat", which should be more understandable
- *uniform-unstick*: warn if a unit belongs to a squad from a different site (can happen with migrants from previous forts)
- *warn-stranded*: center the screen on the unit when you select one in the list
- *work-now*: now saves its enabled status with the fort
- ***zone*:**
  - add include/only/exclude filter for juveniles to the pasture/pit/cage/restraint assignment screen
  - show geld status and custom profession (if set, it's the lower editable line in creature description) in pasture/pit/cage/restraint assignment screen

Removed

- *channel-safely*: (temporarily) removed due to stability issues with the underlying DF API
- `persist-table`: replaced by new `dfhack.persistent` API

API

- New plugin API for saving and loading persistent data. See plugins/examples/skeleton.cpp and plugins/examples/persistent_per_save_example.cpp for details
- Plugin ABI (binary interface) version bump! Any external plugins must be recompiled against this version of DFHack source code in order to load.
- `capitalize_string_words`: new `MiscUtils` function, returns string with all words capitalized
- `Constructions::designateRemove`: no longer designates the non-removable "pseudo" construtions that represent the top of walls
- `grab_token_string_pos`: new `MiscUtils` function, used for parsing tokens
- `Items`: add item melting logic `canMelt(item)`, `markForMelting(item)`, and `cancelMelting(item)`
- **`Persistence`:**
  - persistent keys are now namespaced by an entity_id (e.g. a player fort site ID)
  - data is now stored one file per entity ID (plus one for the global world) in the DF savegame directory
- `random_index`, `vector_get_random`: new `MiscUtils` functions, for getting a random entry in a vector
- `Units.isDanger`: now returns true for agitated wildlife
- **`World`:**
  - `GetCurrentSiteId()` returns the loaded fort site ID (or -1 if no site is loaded)
  - `IsSiteLoaded()` check to detect if a site (e.g. a player fort) is active (as opposed to the world or a map)
  - `AddPersistentData` and related functions replaced with `AddPersistentSiteData` and `AddPersistentWorldData` equivalents

Lua

- `dfhack.capitalizeStringWords`: new function, returns string with all words capitalized
- `dfhack.isSiteLoaded`: returns whether a site (e.g. a player fort) is loaded
- `dfhack.items`: access to `canMelt(item)`, `markForMelting(item)`, and `cancelMelting(item)` from `Items` module
- `dfhack.persistent`: new, table-driven API for easier world- and site-associated persistent storage. See the Lua API docs for details.
- `dfhack.world.getCurrentSite`: returns the `df.world_site` instance of the currently loaded fort
- `widgets.Divider`: linear divider to split an existing frame; configurable T-junction edges and frame style matching

Structures

- `alert_button_announcement_id`: now int32_t vector (contains report ids)
- `announcement_alertst`: defined
- `announcement_alert_type`: enum defined
- `announcement_type`: added `alert_type` enum attribute
- `feature_init_flags`: more enum values defined
- `markup_text_boxst`: updated based on information from Bay12
- `markup_text_linkst`, `markup_text_wordst`, `script_environmentst`: defined
- `occupation`: realigned
- `plotinfost`: `unk23c8_flags` renamed to `flags`, updated based on information from Bay12
- `service_orderst`: type defined
- `service_order_type`: enum defined
- `soundst`: defined
- `viewscreen_choose_start_sitest`: fix structure of warning flags – convert series of bools to a proper bitmask
- `world_raws`: `unk_v50_1`, `unk_v50_2`, `unk_v50_3` renamed to `text_set`, `music`, `sound`

Documentation

- DFHack developer's guide updated, with refreshed *Architecture diagrams*
- UTF-8 text in tool docs is now properly displayed in-game in *gui/launcher* (assuming that it can be converted to cp-437)
- *Installing*: Add installation instructions for wineskin on Mac
- *DFHack modding guide*: Add examples for script-only and blueprint-only mods that you can upload to DF's Steam Workshop

8.4.12 DFHack 50.11-r4

New Tools

- *build-now*: (reinstated) instantly complete unsuspended buildings that are ready to be built

Fixes

- RemoteServer: don't shut down the socket prematurely, allowing continuing connections from, for example, dfhack-run
- *buildingplan*: fix choosing the wrong mechanism (or something that isn't a mechanism) when linking a lever and manually choosing a mechanism, but then canceling the selection
- *combine*: prevent stack sizes from growing beyond quantities that you would normally see in vanilla gameplay
- *gui/design*: Center dragging shapes now track the mouse correctly
- ***sort*:**
  - fix potential crash when exiting and re-entering a creatures subtab with a search active
  - prevent keyboard keys from affecting the UI when search is active and multiple keys are hit at once
- *tailor*: fix corner case where existing stock was being ignored, leading to over-ordering

Misc Improvements

- ***buildingplan*:**
  - save magma safe mechanisms for when magma safety is requested when linking levers and pressure plates to targets
  - when choosing mechanisms for linking levers/pressure plates, filter out unreachable mechanisms
- *caravan*: enable searching within containers in trade screen when in "trade bin with contents" mode
- *sort*: when searching on the Tasks tab, also search the names of the things the task is associated with, such as the name of the stockpile that an item will be stored in

8.4.13 DFHack 50.11-r3

New Tools

- *burrow*: (reinstated) automatically expand burrows as you dig
- *sync-windmills*: synchronize or randomize movement of active windmills
- *trackstop*: (reimplemented) integrated overlay for changing track stop and roller settings after construction

New Features

- *buildingplan*: allow specific mechanisms to be selected when linking levers or pressure plates
- *burrow*: integrated 3d box fill and 2d/3d flood fill extensions for burrow painting mode
- *fix/dead-units*: gained ability to scrub dead units from burrow membership lists
- *gui/design*: show selected dimensions next to the mouse cursor when designating with vanilla tools, for example when painting a burrow or designating digging
- *prospector*: can now give you an estimate of resources from the embark screen. hover the mouse over a potential embark area and run *prospector*.
- *quickfort*: new `burrow` blueprint mode for designating or manipulating burrows
- *sort*: military and burrow membership filters for the burrow assignment screen
- *unforbid*: now ignores worn and tattered items by default (X/XX), use -X to bypass

Fixes

- RemoteServer: continue to accept connections as long as the listening socket is valid instead of closing the socket after the first disconnect
- *buildingplan*: overlay and filter editor gui now uses ctrl-d to delete the filter to avoid conflict with increasing the filter's minimum quality (shift-x)
- *caravan*: price of vermin swarms correctly adjusted down. a stack of 10000 bees is worth 10, not 10000
- *emigration*: fix clearing of work details assigned to units that leave the fort
- *gui/unit-syndromes*: show the syndrome names properly in the UI
- *sort*: when filtering out already-established temples in the location assignment screen, also filter out the "No specific deity" option if a non-denominational temple has already been established
- *stockpiles*: hide configure and help buttons when the overlay panel is minimized
- *tailor*: fix crash on Linux where scanned unit is wearing damaged non-clothing (e.g. a crown)

Misc Improvements

- ***buildingplan***:
  - display how many items are available on the planner panel
  - make it easier to build single-tile staircases of any shape (up, down, or up/down)
- *Dreamfort*: Inside+ and Clearcutting burrows now automatically created and managed
- ***sort***:
  - allow searching by profession on the squad assignment page
  - add search for places screens
  - add search for work animal assignment screen; allow filtering by miltary/squad/civilian/burrow
  - on the squad assignment screen, make effectiveness and potential ratings use the same scale so effectiveness is always less than or equal to potential for a given unit. this way you can also tell when units are approaching their maximum potential

  - new overlay on the animal assignment screen that shows how many work animals each visible unit already has assigned to them

- *warn-stranded*: don't warn for units that are temporarily on unwalkable tiles (e.g. as they pass under a waterfall)

Removed

- ***gui/control-panel*:**
  - removed always-on system services from the `System` tab: *buildingplan*, *confirm*, *logistics*, and *overlay*. The base services should not be turned off by the player. Individual confirmation prompts can be managed via *gui/confirm*, and overlays (including those for *buildingplan* and *logistics*) are managed on the control panel `Overlays` tab.
  - removed *autolabor* from the `Fort` and `Autostart` tabs. The tool does not function correctly with the new labor types, and is causing confusion. You can still enable *autolabor* from the commandline with `enable autolabor` if you understand and accept its limitations.

API

- `Buildings::completebuild`: used to link a newly created building into the world
- `Burrows::setAssignedUnit`: now properly handles inactive burrows
- `Gui::getMousePos`: now takes an optional `allow_out_of_bounds` parameter so coordinates can be returned for mouse positions outside of the game map (i.e. in the blank space around the map)
- `Gui::revealInDwarfmodeMap`: gained `highlight` parameter to control setting the tile highlight on the zoom target
- `Maps::getWalkableGroup`: get the walkability group of a tile
- `Units::getReadableName`: now returns the *untranslated* name

Lua

- `dfhack.buildings.completebuild`: expose new module API
- `dfhack.gui.getMousePos`: support new optional `allow_out_of_bounds` parameter
- `dfhack.gui.revealInDwarfmodeMap`: gained `highlight` parameter to control setting the tile highlight on the zoom target
- `dfhack.maps.getWalkableGroup`: get the walkability group of a tile
- `gui.FRAME_THIN`: a panel frame suitable for floating tooltips

Structures

- `burrow`: add new graphics mode texture and color fields
- `job_item_flags3`: identify additional flags

Documentation

- Document the Lua API for the `dfhack.world` module

8.4.14 DFHack 50.11-r2

New Tools

- *add-recipe*: (reinstated) add reactions to your civ (e.g. for high boots if your civ didn't start with the ability to make high boots)
- *burial*: (reinstated) create tomb zones for unzoned coffins
- *fix/corrupt-jobs*: prevents crashes by automatically removing corrupted jobs
- *preserve-tombs*: keep tombs assigned to units when they die
- *spectate*: (reinstated) automatically follow dwarves, cycling among interesting ones

New Scripts

- *warn-stranded*: new repeatable maintenance script to check for stranded units, similar to *warn-starving*

New Features

- *burial*: new options to configure automatic burial and limit scope to the current z-level
- ***drain-aquifer*:**
  - gained ability to drain just above or below a certain z-level
  - new option to drain all layers except for the first N aquifer layers, in case you want some aquifer layers but not too many
- *gui/control-panel*: `drain-aquifer --top 2` added as an autostart option
- *logistics*: `automelt` now optionally supports melting masterworks; click on gear icon on *stockpiles* overlay frame
- ***sort*:**
  - new search widgets for Info panel tabs, including all "Creatures" subtabs, all "Objects" subtabs, "Tasks", candidate assignment on the "Noble" subtab, and the "Work details" subtab under "Labor"
  - new search and filter widgets for the "Interrogate" and "Convict" screens under "Justice"
  - new search widgets for location selection screen (when you're choosing what kind of guildhall or temple to dedicate)
  - new search widgets for burrow assignment screen and other unit assignment dialogs
  - new search widgets for artifacts on the world/raid screen
  - new search widgets for slab engraving menu; can filter for only units that need a slab to prevent rising as a ghost
- *stocks*: hotkey for collapsing all categories on stocks screen

Fixes

- ***buildingplan*:**
  - remove bars of ash, coal, and soap as valid building materials to match v50 rules
  - fix incorrect required items being displayed sometimes when switching the planner overlay on and off
- *dwarfvet*: fix invalid job id assigned to `Rest` job, which could cause crashes on reload
- *full-heal*: fix removal of corpse after resurrection
- *gui/sandbox*: fix scrollbar moving double distance on click
- *hide-tutorials*: fix the embark tutorial prompt sometimes not being skipped
- *sort*: don't count mercenaries as appointed officials in the squad assignment screen
- *suspendmanager*: fix errors when constructing near the map edge
- *toggle-kbd-cursor*: clear the cursor position when disabling, preventing the game from sometimes jumping the viewport around when cursor keys are hit
- ***zone*:**
  - races without specific child or baby names will now get generic child/baby names instead of an empty string
  - don't show animal assignment link for cages and restraints linked to dungeon zones (which aren't normally assignable)

Misc Improvements

- Help icons added to several complex overlays. clicking the icon runs *gui/launcher* with the help text in the help area
- ***buildingplan*:**
  - support filtering cages by whether they are occupied
  - show how many items you need to make when planning buildings
- *gui/gm-editor*: for fields with primitive types, change from click to edit to click to select, double-click to edit. this should help prevent accidental modifications to the data and make hotkeys easier to use (since you have to click on a data item to use a hotkey on it)
- *gui/overlay*: filter overlays by current context so there are fewer on the screen at once and you can more easily click on the one you want to reposition
- *orders*: `recheck` command now only resets orders that have conditions that can be rechecked
- *overlay*: allow `overlay_onupdate_max_freq_seconds` to be dynamically set to 0 for a burst of high-frequency updates
- *prioritize*: refuse to automatically prioritize dig and smooth/carve job types since it can break the DF job scheduler; instead, print a suggestion that the player use specialized units and vanilla designation priorities
- *quickfort*: now allows constructions to be built on top of constructed floors and ramps, just like vanilla. however, to allow blueprints to be safely reapplied to the same area, for example to fill in buildings whose constructions were canceled due to lost items, floors will not be rebuilt on top of floors and ramps will not be rebuilt on top of ramps
- *sort*: added help button for squad assignment search/filter/sort
- *tailor*: now adds to existing orders if possilbe instead of creating new ones

- *zone*: animals trained for war or hunting are now labeled as such in animal assignment screens

Removed

- `FILTER_FULL_TEXT`: moved from `gui.widgets` to `utils`; if your full text search preference is lost, please reset it in *gui/control-panel*

API

- added `Items::getCapacity`, returns the capacity of an item as a container (reverse-engineered), needed for *combine*

Lua

- added `dfhack.items.getCapacity` to expose the new module API
- added `GRAY` color aliases for `GREY` colors
- `utils.search_text`: text search routine (generalized from internal `widgets.FilteredList` logic)

Structures

- add new globals: `translate_name`, `buildingst_completebuild`
- `artifact_rumor_locationst`: defined
- `viewscreen_worldst`: defined types for `view_mode` and `artifacts_arl` fields
- `world_view_mode_type`: defined

Documentation

- unavailable tools are no longer listed in the tag indices in the online docs

8.4.15 DFHack 50.11-r1

New Tools

- *startdwarf*: (reinstated) set number of starting dwarves
- *tubefill*: (reinstated) replenishes mined-out adamantine

New Features

- A new searchable, sortable, filterable dialog for selecting items for display on pedestals and display cases
- *startdwarf*: overlay scrollbar so you can scroll through your starting dwarves if they don't all fit on the screen

Fixes

- EventManager: Unit death event no longer misfires on units leaving the map
- *autolabor*: ensure vanilla work details are reinstated when the fort or the plugin is unloaded
- *suspendmanager*: fixed a bug where floor grates, bars, bridges etc. wouldn't be recognised as walkable, leading to unnecessary suspensions in certain cases.
- `dfhack.TranslateName()`: fixed crash on certain invalid names, which affected *warn-starving*

Misc Improvements

- **EventManager:**
  - guard against potential iterator invalidation if one of the event listeners were to modify the global data structure being iterated over
  - for `onBuildingCreatedDestroyed` events, changed firing order of events so destroyed events come before created events
- *devel/inspect-screen*: display total grid size for UI and map layers
- ***dig*:**
  - designate only visible tiles by default, and use "auto" dig mode for following veins
  - added options for designating only current z-level, this z-level and above, and this z-level and below
- ***hotkeys*:**
  - make the DFHack logo brighten on hover in ascii mode to indicate that it is clickable
  - use vertical bars instead of "!" symbols for the DFHack logo in ascii mode to make it easier to read
- *suspendmanager*: now suspends constructions that would cave-in immediately on completion

Lua

- mouse key events are now aligned with internal DF semantics: `_MOUSE_L` indicates that the left mouse button has just been pressed and `_MOUSE_L_DOWN` indicates that the left mouse button is being held down. similarly for `_MOUSE_R` and `_MOUSE_M`. 3rd party scripts may have to adjust.

Structures

- add new global: `start_dwarf_count`

8.4.16 DFHack 50.10-r1

Fixes

- 'fix/general-strike: fix issue where too many seeds were getting planted in farm plots
- Linux launcher: allow Steam Overlay and game streaming to function
- *autobutcher*: don't ignore semi-wild units when marking units for slaughter

Misc Improvements

- 'sort': Improve combat skill scale thresholds

8.4.17 DFHack 50.09-r4

New Features

- *dig*: new overlay for ASCII mode that visualizes designations for smoothing, engraving, carving tracks, and carving fortifications

Fixes

- *buildingplan*: make the construction dimensions readout visible again
- *gui/mod-manager*: don't continue to display overlay after the raws loading progress bar appears
- *seedwatch*: fix a crash when reading data saved by very very old versions of the plugin

Misc Improvements

- *autofish*: changed `--raw` argument format to allow explicit setting to on or off
- *caravan*: move goods to depot screen can now see/search/trade items inside of barrels and pots
- *gui/launcher*: show tagged tools in the autocomplete list when a tag name is typed
- ***sort*:**
  - add sort option for training need on squad assignment screen
  - filter mothers with infants, units with weak mental fortitude, and critically injured units on the squad assignment screen
  - display a rating relative to the current sort order next to the visible units on the squad assignment screen

API

- *overlay*: overlay widgets can now declare a `version` attribute. changing the version of a widget will reset its settings to defaults. this is useful when changing the overlay layout and old saved positions will no longer be valid.

Lua

- `argparse.boolean`: convert arguments to lua boolean values.

Structures

- Identified a number of previously anonymous virtual methods in `itemst`

Documentation

- add instructions for downloading development builds to the `Installing` page

8.4.18 DFHack 50.09-r3

New Tools

- *devel/scan-vtables*: scan and dump likely vtable addresses (for memory research)
- *hide-interface*: hide the vanilla UI elements for clean screenshots or laid-back fortress observing
- *hide-tutorials*: hide the DF tutorial popups; enable in the System tab of *gui/control-panel*
- *set-orientation*: tinker with romantic inclinations (reinstated from back catalog of tools)

New Features

- *buildingplan*: one-click magma/fire safety filter for planned buildings
- *exportlegends*: new overlay that integrates with the vanilla "Export XML" button. Now you can generate both the vanilla export and the extended data export with a single click!
- *sort*: search, sort, and filter for squad assignment screen
- *zone*: advanced unit assignment screens for cages, restraints, and pits/ponds

Fixes

- **Core:**
  - reload scripts in mods when a world is unloaded and immediately loaded again
  - fix text getting added to DFHack text entry widgets when Alt- or Ctrl- keys are hit
- *autobutcher*: fix `ticks` commandline option incorrectly rejecting positive integers as valid values
- *buildingplan*: ensure selected barrels and buckets are empty (or at least free of lye and milk) as per the requirements of the building
- ***caravan*:**
  - corrected prices for cages that have units inside of them
  - correct price adjustment values in trade agreement details screen
  - apply both import and export trade agreement price adjustments to items being both bought or sold to align with how vanilla DF calculates prices
  - cancel any active TradeAtDepot jobs if all caravans are instructed to leave
- ***emigration*:**
  - fix errors loading forts after dwarves assigned to work details or workshops have emigrated
  - fix citizens sometimes "emigrating" to the fortress site

- *fix/retrieve-units*: fix retrieved units sometimes becoming duplicated on the map
- *gui/launcher*, *gui/gm-editor*: recover gracefully when the saved frame position is now offscreen
- *gui/sandbox*: correctly load equipment materials in modded games that categorize non-wood plants as wood
- *orders*: prevent import/export overlay from appearing on the create workorder screen
- *quickfort*: cancel old dig jobs that point to a tile when a new designation is applied to the tile
- *seedwatch*: ignore unplantable tree seeds
- *starvingdead*: ensure sieges end properly when undead siegers starve
- ***suspendmanager*:**
  - Fix the overlay enabling/disabling *suspendmanager* unexpectedly
  - improve the detection on "T" and "+" shaped high walls
- *tailor*: remove crash caused by clothing items with an invalid `maker_race`
- `dialogs.MessageBox`: fix spacing around scrollable text

Misc Improvements

- Surround DFHack-specific UI elements with square brackets instead of red-yellow blocks for better readability
- *autobutcher*: don't mark animals for butchering if they are already marked for some kind of training (war, hunt)
- *caravan*: optionally display items within bins in bring goods to depot screen
- *createitem*: support creating items inside of bags
- *devel/lsmem*: added support for filtering by memory addresses and filenames
- *gui/design*: change "auto commit" hotkey from `c` to `Alt-c` to avoid conflict with the default keybinding for z-level down
- ***gui/gm-editor*:**
  - hold down shift and right click to exit, regardless of how many substructures deep you are
  - display in the title bar whether the editor window is scanning for live updates
- *gui/liquids*: support removing river sources by converting them into stone floors
- *gui/quickfort*: blueprint details screen can now be closed with Ctrl-D (the same hotkey used to open the details)
- *hotkeys*: don't display DFHack logo in legends mode since it covers up important interface elements. the Ctrl-Shift-C hotkey to bring up the menu and the mouseover hotspot still function, though.
- *quickfort*: linked stockpiles and workshops can now be specified by ID instead of only by name. this is mostly useful when dynamically generating blueprints and applying them via the *quickfort* API
- *sort*: animals are now sortable by race on the assignment screens
- *suspendmanager*: display a different color for jobs suspended by suspendmanager

API

- *RemoteFortressReader*: add a `force_reload` option to the GetBlockList RPC API to return blocks regardless of whether they have changed since the last request
- `Gui`: `getAnyStockpile` and `getAnyCivzone` (along with their `getSelected` variants) now work through layers of ZScreens. This means that they will still return valid results even if a DFHack tool window is in the foereground.
- `Items::getValue()`: remove `caravan_buying` parameter since the identity of the selling party doesn't actually affect the item value
- `Units`: new animal propery check functions `isMarkedForTraining(unit)`, `isMarkedForTaming(unit)`, `isMarkedForWarTraining(unit)`, and `isMarkedForHuntTraining(unit)`

Lua

- `dfhack.gui`: new `getAnyCivZone` and `getAnyStockpile` functions; also behavior of `getSelectedCivZone` and `getSelectedStockpile` functions has changes as per the related API notes
- `dfhack.items.getValue()`: remove `caravan_buying` param as per C++ API change
- `dfhack.screen.readTile()`: now populates extended tile property fields (like `top_of_text`) in the returned `Pen` object
- `dfhack.units`: new animal propery check functions `isMarkedForTraining(unit)`, `isMarkedForTaming(unit)`, `isMarkedForWarTraining(unit)`, and `isMarkedForHuntTraining(unit)`
- `new()`: improved error handling so that certain errors that were previously uncatchable (creating objects with members with unknown vtables) are now catchable with `pcall()`
- `widgets.BannerPanel`: panel with distinctive border for marking DFHack UI elements on otherwise vanilla screens
- `widgets.Panel`: new functions to override instead of setting corresponding properties (useful when subclassing instead of just setting attributes): `onDragBegin`, `onDragEnd`, `onResizeBegin`, `onResizeEnd`

Structures

- Added `global_table` global and corresponding `global_table_entry` type
- `help_context_type`: fix typo in enum name: `EMBARK_TUTORIAL_CHICE` -> `EMBARK_TUTORIAL_CHOICE`
- `plotinfo`: name the fields related to tutorial popups
- `viewscreen_legendsst`: realign structure
- `viewscreen_new_arenast`: added (first appeared in 50.06)

8.4.19 DFHack 50.09-r2

New Plugins

- *3dveins*: reinstated for v50, this plugin replaces vanilla DF's blobby vein generation with veins that flow smoothly and naturally between z-levels
- *dig*: new `dig.asciiwarmdamp` overlay that highlights warm and damp tiles when in ASCII mode. there is no effect in graphics mode since the tiles are already highlighted there
- *dwarfvet*: reinstated and updated for v50's new hospital mechanics; allow your animals to have their wounds treated at hospitals
- *zone*: new searchable, sortable, filterable screen for assigning units to pastures

New Scripts

- *caravan*: new trade screen UI replacements for bringing goods to trade depot and trading
- *fix/empty-wheelbarrows*: new script to empty stuck rocks from all wheelbarrows on the map

Fixes

- Fix extra keys appearing in DFHack text boxes when shift (or any other modifier) is released before the other key you were pressing
- *gui/autodump*: when "include items claimed by jobs" is on, actually cancel the job so the item can be teleported
- *gui/create-item*: when choosing a citizen to create the chosen items, avoid choosing a dead citizen
- *gui/gm-unit*: fix commandline processing when a unit id is specified
- ***logistics***:
  - don't autotrain domestic animals brought by invaders (they'll get attacked by friendly creatures as soon as you let them out of their cage)
  - don't bring trade goods to depot if the only caravans present are tribute caravans
  - fix potential crash when removing stockpiles or turning off stockpile features
- ***suspendmanager***:
  - take in account already built blocking buildings
  - don't consider tree branches as a suitable access path to a building

Misc Improvements

- Dreamfort: give noble suites double-thick walls and add apartment doors
- Suppress DF keyboard events when a DFHack keybinding is matched. This prevents, for example, a backtick from appearing in a textbox as text when you launch *gui/launcher* from the backtick keybinding.
- *autonick*: add more variety to nicknames based on famous literary dwarves
- *gui/unit-syndromes*: make lists searchable
- *logistics*: bring an autotraded bin to the depot if any item inside is tradeable instead of marking all items within the bin as untradeable if any individual item is untradeable

- *quickfort*: blueprint libraries are now moddable – add a `blueprints/` directory to your mod and they'll show up in *quickfort* and *gui/quickfort*!
- *stockpiles*: include exotic pets in the "tameable" filter
- *suspendmanager*: display the suspension reason when viewing a suspended building
- `widgets.EditField`: DFHack edit fields now support cut/copy/paste with the system clipboard with Ctrl-X/Ctrl-C/Ctrl-V

API

- `Items::markForTrade()`, `Items::isRequestedTradeGood()`, `Items::getValue`: see Lua notes below
- `Units::getUnitByNobleRole`, `Units::getUnitsByNobleRole`: unit lookup API by role

Internals

- Price calculations fixed for many item types

Lua

- `dfhack.items.getValue`: gained optional `caravan` and `caravan_buying` parameters for prices that take trader races and agreements into account
- `dfhack.items.isRequestedTradeGood`: discover whether an item is named in a trade agreement with an active caravan
- `dfhack.items.markForTrade`: mark items for trade
- `dfhack.units.getUnitByNobleRole`, `dfhack.units.getUnitsByNobleRole`: unit lookup API by role
- `widgets.TextButton`: wraps a `HotkeyLabel` and decorates it to look more like a button

Structures

- `build_req_choicest`: realign structure and fix vmethods
- `squad_orderst`: fix vmethods

Documentation

- *misery*: rewrite the documentation to clarify the actual effects of the plugin

8.4.20 DFHack 50.09-r1

Misc Improvements

- *caravan*: new overlay for selecting all/none on trade request screen
- *suspendmanager*: don't suspend constructions that are built over open space

Internals

- Core: update SDL interface from SDL1 to SDL2

Structures

- `tiletype_shape`: changed RAMP_TOP and ENDLESS_PIT to not walkable to reflect how scripts actually need these types to be treated

8.4.21 DFHack 50.08-r4

New Plugins

- *logistics*: automatically mark and route items or animals that come to monitored stockpiles. options are toggleable on an overlay that comes up when you have a stockpile selected.

Fixes

- *buildingplan*: don't include artifacts when max quality is masterful
- *dig-now*: clear item occupancy flags for channeled tiles that had items on them
- *emigration*: reassign home site for emigrating units so they don't just come right back to the fort
- *gui/create-item*: allow blocks to be made out of wood when using the restrictive filters
- *gui/liquids*: ensure tile temperature is set correctly when painting water or magma
- ***gui/quickfort*:**
  - allow traffic designations to be applied over buildings
  - protect against meta blueprints recursing infinitely if they include themselves
- *gui/sandbox*: allow creatures that have separate caste-based graphics to be spawned (like ewes/rams)
- *RemoteFortressReader*: fix a crash with engravings with undefined images
- *workorder*: prevent `autoMilkCreature` from over-counting milkable animals, which was leading to cancellation spam for the MilkCreature job

Misc Improvements

- **Blueprint library:**
  - dreamfort: full rewrite and update for DF v50
  - pump_stack: updated walkthrough and separated dig and channel steps so boulders can be cleared
  - aquifer_tap: updated walkthrough
- *autonick*: additional nicknames based on burrowing animals, colours, gems, and minerals
- *combine*: reduce max different stacks in containers to 30 to prevent contaners from getting overfull
- *dig-now*: can now handle digging obsidian that has been formed from magma and water
- *gui/autodump*: add option to clear the `trader` flag from teleported items, allowing you to reclaim items dropped by merchants

- ***gui/control-panel*:**
  - add some popular startup configuration commands for *autobutcher* and *autofarm*
  - add option for running *fix/blood-del* on new forts (enabled by default)
- ***gui/quickfort*:**
  - adapt "cursor lock" to mouse controls so it's easier to see the full preview for multi-level blueprints before you apply them
  - only display post-blueprint messages once when repeating the blueprint up or down z-levels
- *gui/sandbox*: when creating citizens, give them names appropriate for their races
- ***orders*:**
  - only display import/export/sort/clear panel on main orders screen
  - refine order conditions for library orders to reduce cancellation spam
- *prioritize*: add wild animal management tasks and lever pulling to the default list of prioritized job types
- *quickfort*: significant rewrite for DF v50! now handles zones, locations, stockpile configuration, hauling routes, and more
- *stockpiles*: added `barrels`, `organic`, `artifacts`, and `masterworks` stockpile presets
- ***suspendmanager*:**
  - now suspends construction jobs on top of floor designations, protecting the designations from being erased
  - suspend blocking jobs when building high walls or filling corridors
- *workorder*: reduce existing orders for automatic shearing and milking jobs when animals are no longer available

Removed

- *gui/automelt*: replaced by an overlay panel that appears when you click on a stockpile

Structures

- `abstract_building_libraryst`: initialize unknown variables as DF does
- `misc_trait_type`: realign

Documentation

- *Quickfort blueprint library*: update Dreamfort screenshots and links, add `aquifer_tap` screenshot

8.4.22 DFHack 50.08-r3

Fixes

- Fix crash for some players when they launch DF outside of the Steam client

8.4.23 DFHack 50.08-r2

New Plugins

- *add-spatter*: (reinstated) allow mods to add poisons and magical effects to weapons
- *changeitem*: (reinstated) change item material, quality, and subtype
- *createitem*: (reinstated) create arbitrary items from the command line
- *deramp*: (reinstated) removes all ramps designated for removal from the map
- *flows*: (reinstated) counts map blocks with flowing liquids
- *lair*: (reinstated) mark the map as a monster lair (this avoids item scatter when the fortress is abandoned)
- *luasocket*: (reinstated) provides a Lua API for accessing network sockets
- *work-now*: (reinstated, renamed from `workNow`) prevent dwarves from wandering aimlessly with "No job" after completing a task

New Scripts

- *assign-minecarts*: (reinstated) quickly assign minecarts to hauling routes
- *diplomacy*: view or alter diplomatic relationships
- *exportlegends*: (reinstated) export extended legends information for external browsing
- *fix/stuck-instruments*: fix instruments that are attached to invalid jobs, making them unusable. turn on automatic fixing in *gui/control-panel* in the `Maintenance` tab.
- *gui/autodump*: point and click item teleportation and destruction interface (available only if `armok` tools are shown)
- *gui/mod-manager*: automatically restore your list of active mods when generating new worlds
- *gui/sandbox*: creation interface for units, trees, and items (available only if `armok` tools are shown)
- *light-aquifers-only*: (reinstated) convert heavy aquifers to light
- *modtools/create-item*: (reinstated) commandline and API interface for creating items
- *necronomicon*: search fort for items containing the secrets of life and death

Fixes

- DFHack screen backgrounds now use appropriate tiles in DF Classic
- RemoteServer: fix crash on malformed json in `dfhack-config/remote-server.json`
- *autolabor*: work detail override warning now only appears on the work details screen
- *deathcause*: fix incorrect weapon sometimes being reported
- *gui/create-item*: allow armor to be made out of leather when using the restrictive filters
- *gui/design*: Fix building and stairs designation
- ***quickfort*:**
  - properly allow dwarves to smooth, engrave, and carve beneath walkable tiles of buildings
  - fixed detection of tiles where machines are allowed (e.g. water wheels *can* be built on stairs if there is a machine support nearby)
  - fixed rotation of blueprints with carved track tiles
- *RemoteFortressReader*: ensured names are transmitted in UTF-8 instead of CP437

Misc Improvements

- Core: new commandline flag/environment var: pass `--disable-dfhack` on the Dwarf Fortress commandline or specify `DFHACK_DISABLE=1` in the environment to disable DFHack for the current session.
- Dreamfort: improve traffic patterns throughout the fortress
- Settings: recover gracefully when settings files become corrupted (e.g. by DF CTD)
- **Window behavior:**
  - non-resizable windows now allow dragging by their frame edges by default
  - if you have multiple DFHack tool windows open, scrolling the mouse wheel while over an unfocused window will focus it and raise it to the top
- *autodump*: no longer checks for a keyboard cursor before executing, so `autodump destroy` (which doesn't require a cursor) can still function
- *gui/autodump*: fort-mode keybinding: Ctrl-H (when `armok` tools are enabled in *gui/control-panel*)
- *gui/blueprint*: recording of stockpile layouts and categories is now supported. note that detailed stockpile configurations will *not* be saved (yet)
- *gui/control-panel*: new preference for whether filters in lists search for substrings in the middle of words (e.g. if set to true, then "ee" will match "steel")
- *gui/create-item*: ask for number of items to spawn by default
- *gui/design*: Improved performance for drawing shapes
- ***gui/gm-editor*:**
  - when passing the `--freeze` option, further ensure that the game is frozen by halting all rendering (other than for DFHack tool windows)
  - Alt-A now enables auto-update mode, where you can watch values change live when the game is unpaused
- ***gui/quickfort*:**

  - blueprints that designate items for dumping/forbidding/etc. no longer show an error highlight for tiles that have no items on them
  - place (stockpile layout) mode is now supported. note that detailed stockpile configurations were part of query mode and are not yet supported
  - you can now generate manager orders for items required to complete bluerpints
- *light-aquifers-only*: now available as a fort Autostart option in *gui/control-panel*. note that it will only appear if "armok" tools are configured to be shown on the Preferences tab.
- *orders*: update orders in library for prepared meals, bins, archer uniforms, and weapons
- *overlay*: add links to the quickstart guide and the control panel on the DF title screen
- *stockpiles*: allow filtering creatures by tameability

Removed

- *orders*: `library/military_include_artifact_materials` library file removed since recent research indicates that platinum blunt weapons and silver crossbows are not more effective than standard steel. the alternate military orders file was also causing unneeded confusion.

Internals

- `dfhack.internal`: added memory analysis functions: `msizeAddress`, `getHeapState`, `heapTakeSnapshot`, `isAddressInHeap`, `isAddressActiveInHeap`, `isAddressUsedAfterFreeInHeap`, `getAddressSizeInHeap`, and `getRootAddressOfHeapObject`

Lua

- `ensure_keys`: walks a series of keys, creating new tables for any missing values
- `gui`: changed frame naming scheme to `FRAME_X` rather than `X_FRAME`, and added aliases for backwards compatibility. (for example `BOLD_FRAME` is now called `FRAME_BOLD`)
- `overlay.reload()`: has been renamed to `overlay.rescan()` so as not to conflict with the global `reload()` function. If you are developing an overlay, please take note of the new function name for reloading your overlay during development.

Structures

- Removed `steam_mod_manager` and `game_extra` globals. Their contents have been merged back into `game`.
- `abstract_building_contents`: identify fields and flags related to location item counts
- `arena_tree`: identify fields related to tree creation
- `arena_unit`: identify fields related to unit creation
- `mod_headerst`: rename `non_vanilla` flag to `vanilla` to reflect its actual usage
- `profession`: renamed captions `Cheese Maker` to `Cheesemaker`, `Bee Keeper` to `Beekeeper`, and `Bone Setter` to `Bone Doctor`

8.4.24 DFHack 50.08-r1

Fixes

- *autoclothing*: eliminate game lag when there are many inventory items in the fort
- ***buildingplan***:
  - fixed size limit calculations for rollers
  - fixed items not being checked for accessibility in the filter and item selection dialogs
- *deteriorate*: ensure remains of enemy dwarves are properly deteriorated
- *dig-now*: properly detect and complete smoothing designations that have been converted into active jobs
- *suspendmanager*: Fix over-aggressive suspension of jobs that could still possibly be done (e.g. jobs that are partially submerged in water)

Misc Improvements

- ***buildingplan***:
  - planner panel is minimized by default and now remembers minimized state
  - can now filter by gems (for gem windows) and yarn (for ropes in wells)
- *combine*: Now supports ammo, parts, powders, and seeds, and combines into containers
- *deteriorate*: add option to exclude useable parts from deterioration
- ***gui/control-panel***:
  - add preference option for hiding the terminal console on startup
  - add preference option for hiding "armok" tools in command lists
- ***gui/gm-editor***:
  - press `g` to move the map to the currently selected item/unit/building
  - press `Ctrl-D` to toggle read-only mode to protect from accidental changes; this state persists across sessions
  - new `--freeze` option for ensuring the game doesn't change while you're inspecting it
- *gui/launcher*: DFHack version now shown in the default help text
- *gui/prerelease-warning*: widgets are now clickable
- *overlay*: add the DFHack version string to the DF title screen
- `Dwarf Therapist`: add a warning to the Labors screen when Dwarf Therapist is active so players know that changes they make to that screen will have no effect. If you're starting a new embark and nobody seems to be doing anything, check your Labors tab for this warning to see if Dwarf Therapist thinks it is in control (even if it's not running).
- `toggle-kbd-cursor`: add hotkey for toggling the keyboard cursor (Alt-K)
- `version`: add alias to display the DFHack help (including the version number) so something happens when players try to run "version"

Removed

- *title-version*: replaced by an *overlay* widget

Lua

- `gui.ZScreenModal`: ZScreen subclass for modal dialogs
- `widgets.CycleHotkeyLabel`: exposed "key_sep" and "option_gap" attributes for improved stylistic control.
- `widgets.RangeSlider`: new mouse-controlled two-headed slider widget

Structures

- convert `mod_manager` fields to pointers

8.4.25 DFHack 50.07-r1

New Plugins

- *autoslab*: automatically create work orders to engrave slabs for ghostly dwarves
- *faststart*: speeds up the "Loading..." screen so the Main Menu appears faster
- *getplants*: reinstated: designate trees for chopping and shrubs for gathering according to type
- *prospector*: reinstated: get stone, ore, gem, and other tile property counts in fort mode.

New Scripts

- *allneeds*: list all unmet needs sorted by how many dwarves suffer from them.
- *autofish*: auto-manage fishing labors to control your stock of fish
- *combine*: combines stacks of food and plant items.
- *devel/tile-browser*: page through available textures and see their texture ids
- *fix/civil-war*: removes negative relations with own government
- *fix/general-strike*: fix known causes of the general strike bug (contributed by Putnam)
- *fix/protect-nicks*: restore nicknames when DF loses them
- *forbid*: forbid and list forbidden items on the map
- *gui/autochop*: configuration frontend and status monitor for the *autochop* plugin
- *gui/autofish*: GUI config and status panel interface for autofish
- *gui/automelt*: GUI config and status panel interface for automelt
- *gui/civ-alert*: configure and trigger civilian alerts
- *gui/control-panel*: quick access to DFHack configuration
- *gui/design*: digging and construction designation tool with shapes and patterns
- *gui/seedwatch*: GUI config and status panel interface for *seedwatch*
- *gui/suspendmanager*: graphical configuration interface for *suspendmanager*

- *gui/unit-syndromes*: browser for syndrome information
- *makeown*: makes the selected unit a citizen of your fortress
- *suspendmanager*: automatic job suspension management (replaces *autounsuspend*)
- *suspend*: suspends building construction jobs

Fixes

- *autochop*: fixed a crash when processing trees with corrupt data structures (e.g. when a trunk tile fails to fall when the rest of the tree is chopped down)
- *autoclothing*: fixed a crash that can happen when units are holding invalid items.
- *autodump*: changed behaviour to only change `dump` and `forbid` flags if an item is successfully dumped.
- *blueprint*: interpret saplings, shrubs, and twigs as floors instead of walls
- ***build-now*:**
  - now correctly avoids adjusting non-empty tiles above constructions that it builds
  - don't error on constructions that do not have an item attached
- ***buildingplan*:**
  - upright spike traps are now placed extended rather than retracted
  - you can no longer designate constructions on tiles with magma or deep water, mirroring the vanilla restrictions
- *catsplosion*: now only affects live, active units
- *combine*: fix error processing stockpiles with boundaries that extend outside of the map
- *devel/query*: can now properly index vectors in the –table argument
- *dig-now*: fixed multi-layer channel designations only channeling every second layer
- *getplants*: trees are now designated correctly
- *gui/launcher*: tab characters in command output now appear as a space instead of a code page 437 "blob"
- *make-legendary*: "MilitaryUnarmed" option now functional
- *orders*: library/military now sticks to vanilla rules and does not add orders for normally-mood-only platinum weapons. A new library orders file `library/military_include_artifact_materials` is now offered as an alternate `library/military` set of orders that still includes the platinum weapons.
- *quickfort*: allow floor bars, floor grates, and hatches to be placed over all stair types like vanilla allows
- *showmood* properly count required number of bars and cloth when they aren't the main item for the strange mood
- *stockpiles*: fix crash when importing settings for gems from other worlds
- ***tailor*:**
  - properly discriminate between dyed and undyed cloth
  - take queued orders into account when calculating available materials
  - skip units who can't wear clothes
  - identify more available items as available, solving issues with over-production
  - block making clothing sized for toads; make replacement clothing orders use the size of the wearer, not the size of the garment

  - now respects the setting of the "used dyed clothing" standing order toggle

- ***warn-starving*:**
  - no longer warns for dead units
  - no longer warns for enemy and neutral units
- `Buildings::StockpileIterator`: fix check for stockpile items on block boundary.
- `dfhack.job.isSuitableMaterial`: now properly detects lack of fire and magma safety for vulnerable materials with high melting points
- `Units::isFortControlled`: Account for agitated wildlife
- `widgets.WrappedLabel`: no longer resets scroll position when window is moved or resized

Misc Improvements

- Core: hide DFHack terminal console by default when running on Steam Deck
- DFHack tool windows that capture mouse clicks (and therefore prevent you from clicking on the "pause" button) now unconditionally pause the game when they open (but you can still unpause with the keyboard if you want to). Examples of this behavior: *gui/quickfort*, *gui/blueprint*, *gui/liquids*
- Mods: scripts in mods are now automatically added to the DFHack script path. DFHack recognizes two directories in a mod's folder: `scripts_modinstalled/` and `scripts_modactive/`. `scripts_modinstalled/` folders will always be added the script path, regardless of whether the mod is active in a world. `scripts_modactive/` folders will only be added to the script path when the mod is active in the current loaded world.
- Scrollable widgets now react to mouse wheel events when the mouse is over the widget
- the `dfhack-config/scripts/` folder is now searched for scripts by default
- ***autobutcher*:**
  - logs activity to the console terminal instead of making disruptive in-game announcements
  - changed defaults from 5 females / 1 male to 4 females / 2 males so a single unfortunate accident doesn't leave players without a mating pair
  - now immediately loads races available at game start into the watchlist
- *autodump*: new hotkey for `autodump-destroy-here`: Ctrl-H
- ***automelt*:**
  - now allows metal chests to be melted (workaround for DF bug 2493 is no longer needed)
  - stockpile configuration can now be set from the commandline
- *autounsuspend*: now saves its state with your fort
- ***ban-cooking*:**
  - ban announcements are now hidden by default; use new option `--verbose` to show them.
  - report number of items banned.
- ***Quickfort blueprint library*:**
  - library blueprints have moved from `blueprints` to `hack/data/blueprints`

  - player-created blueprints should now go in the `dfhack-config/blueprints` folder. please move your existing blueprints from `blueprints` to `dfhack-config/blueprints`. you don't need to move the library blueprints – those can be safely deleted from the old `blueprints` directory.

- *blueprint*: now writes blueprints to the `dfhack-config/blueprints` directory
- *build-now*: now handles dirt roads and initializes farm plots properly
- ***buildingplan***:
  - can now filter by cloth and silk materials (for ropes)
  - filters and global settings are now ignored when manually choosing items for a building, allowing you to make custom choices independently of the filters that would otherwise be used
  - if *suspendmanager* is running, then planned buildings will be left suspended when their items are all attached. *suspendmanager* will unsuspsend them for construction when it is safe to do so.
  - add option for autoselecting the last manually chosen item (like *automaterial* used to do)
  - entirely new UI for building placement, item selection, and materials filtering!
- *caravan*: add trade screen overlay that assists with seleting groups of items and collapsing groups in the UI
- *channel-safely*: new monitoring for cave-in prevention
- ***combine***:
  - you can select a target stockpile in the UI instead of having to use the keyboard cursor
  - added `--quiet` option for no output when there are no changes
- ***confirm***:
  - adds confirmation for removing burrows via the repaint menu
  - configuration data is now persisted globally.
- *devel/click-monitor*: report on middle mouse button actions
- *devel/inspect-screen*: updated for new rendering semantics and can now also inspect map textures
- *devel/query*: will now search for jobs at the map coordinate highlighted, if no explicit job is highlighted and there is a map tile highlighted
- *devel/visualize-structure*: now automatically inspects the contents of most pointer fields, rather than inspecting the pointers themselves
- *dig-now*: added handling of dig designations that have been converted into active jobs
- *emigration*: now saves its state with your fort
- *enable*: can now interpret aliases defined with the *alias* command
- ***exterminate***:
  - add support for `vaporize` kill method for when you don't want to leave a corpse
  - added drown method. magma and drown methods will now clean up liquids automatically.
- *getplants*: ID values will now be accepted regardless of case
- ***gui/control-panel***:
  - bugfix services are now enabled by default
  - new global hotkey for quick access: Ctrl-Shift-E
  - Now detects overlays from scripts named with capital letters

- ***gui/cp437-table*:**
  - now has larger key buttons and clickable backspace/submit/cancel buttons, making it fully usable on the Steam Deck and other systems that don't have an accessible keyboard
  - dialog is now fully controllable with the mouse, including highlighting which key you are hovering over and adding a clickable backspace button
  - converted to a movable, mouse-enabled window
- ***gui/create-item*:**
  - added ability to spawn 'whole' corpsepieces (every layer of a part)
  - support spawning corpse pieces (e.g. shells) under "body part"
  - added search and filter capabilities to the selection lists
  - added whole corpse spawning alongside corpsepieces. (under "corpse")
- ***gui/gm-editor*:**
  - can now jump to material info objects from a mat_type reference with a mat_index using `i`
  - the key column now auto-fits to the widest key
  - can now open the selected stockpile if run without parameters
  - will now inspect a selected building itself if the building has no current jobs
  - now supports multiple independent data inspection windows
  - now prints out contents of coordinate vars instead of just the type
  - converted to a movable, resizable, mouse-enabled window
- *gui/gm-unit*: converted to a movable, resizable, mouse-enabled window
- ***gui/launcher*:**
  - sped up initialization time for faster window appearance
  - make command output scrollback separate from the help text so players can continue to see the output of the previous command as they type the next one
  - allow double spacebar to pause/unpause the game, even while typing a command
  - clarify what is being shown in the autocomplete list (all commands, autocompletion of partially typed command, or commands related to typed command)
  - support running commands directly from the autocomplete list via double-clicking
  - now supports a smaller, minimal mode. click the toggle in the launcher UI or start in minimal mode via the `Ctrl-Shift-P` keybinding
  - can now be dragged from anywhere on the window body
  - now remembers its size and position between invocations
- *gui/liquids*: interface overhaul, also now allows spawning river sources, setting/adding/removing liquid levels, and cleaning water from being salty or stagnant
- *gui/overlay*: now focuses on repositioning overlay widgets; enabling, disabling, and getting help for overlay widgets has moved to the new *gui/control-panel*
- ***gui/quickcmd*:**
  - now has its own global keybinding for your convenience: Ctrl-Shift-A

  - converted to a movable, resizable, mouse-enabled window
  - commands are now stored globally so you don't have to recreate commands for every fort
- *gui/quickfort*: don't close the window when applying a blueprint so players can apply the same blueprint multiple times more easily
- *hotkeys*: overlay hotspot widget now shows the DFHack logo in graphics mode and "DFHack" in text mode
- *locate-ore*: now only searches revealed tiles by default
- *misery*: now persists state with the fort
- *modtools/spawn-liquid*: sets tile temperature to stable levels when spawning water or magma
- *nestboxes*: now saves enabled state in your savegame
- ***orders*:**
  - add minimize button to overlay panel so you can get it out of the way to read long statue descriptions when choosing a subject in the details screen
  - add option to delete exported files from the import dialog
  - orders plugin functionality is now accessible via an *overlay* widget when the manager orders screen is open
- ***prioritize*:**
  - revise and simplify the default list of prioritized jobs – be sure to tell us if your forts are running noticeably better (or worse!)
  - now automatically starts boosting the default list of job types when enabled
  - now saves its state with your fort
- *quickfort*: now reads player-created blueprints from `dfhack-config/blueprints/` instead of the old `blueprints/` directory. Be sure to move over your personal blueprints to the new directory!
- *rejuvenate*: now takes an –age parameter to choose a desired age.
- *Script paths*: removed "raw" directories from default script paths. now the default locations to search for scripts are `dfhack-config/scripts`, `save/*/scripts`, and `hack/scripts`
- *seedwatch*: now persists enabled state in the savegame, automatically loads useful defaults, and respects reachability when counting available seeds
- *showmood*: now shows the number of items needed for cloth and bars in addition to the technically correct but always confusing "total dimension" (150 per bar or 10,000 per cloth)
- ***stockpiles*:**
  - support applying stockpile configurations with fully enabled categories to stockpiles in worlds other than the one where the configuration was exported from
  - support partial application of a saved config based on dynamic filtering (e.g. disable all tallow in a food stockpile, even tallow from world-specific generated creatures)
  - additive and subtractive modes when applying a second stockpile configuration on top of a first
  - write player-exported stockpile configurations to the `dfhack-config/stockpiles` folder. If you have any stockpile configs in other directories, please move them to that folder.
  - now includes a library of useful stockpile configs (see docs for details)
- *stonesense*: added an `INVERT_MOUSE_Z` option to invert the mouse wheel direction
- ***stripcaged*:**

  - added `--skip-forbidden` option for greater control over which items are marked for dumping
  - items that are marked for dumping are now automatically unforbidden (unless `--skip-forbidden` is set)
- *tailor*: add support for adamantine cloth (off by default); improve logging
- ***troubleshoot-item*:**
  - output as bullet point list with indenting, with item description and ID at top
  - reports on items that are hidden, artifacts, in containers, and held by a unit
  - reports on the contents of containers with counts for each contained item type
- *unforbid*: avoids unforbidding unreachable and underwater items by default
- ***unsuspend*:**
  - overlay now displays different letters for different suspend states so they can be differentiated in graphics mode (P=planning, x=suspended, X=repeatedly suspended)
  - overlay now shows a marker all the time when in graphics mode. ascii mode still only shows when paused so that you can see what's underneath.
- `init.d`: directories have moved from the `raw` subfolder (which no longer exists) to the root of the main DF folder or a savegame folder

Removed

- Ruby is no longer a supported DFHack scripting language
- *autohauler*: no plans to port to v50, as it just doesn't make sense with the new work detail system
- *automaterial*: all functionality has been merged into *buildingplan*
- *autounsuspend*: replaced by *suspendmanager*
- *combine-drinks*: replaced by *combine*
- *combine-plants*: replaced by *combine*
- *create-items*: replaced by *gui/create-item* `--multi`
- *show-unit-syndromes*: replaced by *gui/unit-syndromes*; html export is no longer supported
- `fix-job-postings` from the *workflow* plugin is now obsolete since affected savegames can no longer be loaded

API

- Gui focus strings will no longer get the "dfhack/" prefix if the string "dfhack/" already exists in the focus string
- Units module: added new predicates for `isGeldable()`, `isMarkedForGelding()`, and `isPet()`
- *overlay*: overlay widgets can now specify a default enabled state if they are not already set in the player's overlay config file
- `Buildings::containsTile()`: no longer takes a `room` parameter since that's not how rooms work anymore. If the building has extents, the extents will be checked. otherwise, the result just depends on whether the tile is within the building's bounding box.
- `Gui::any_civzone_hotkey`, `Gui::getAnyCivZone`, `Gui::getSelectedCivZone`: new functions to operate on the new zone system
- `Gui::getDFViewscreen`: returns the topmost underlying DF viewscreen

- `Gui::getDwarfmodeDims`: now only returns map viewport dimensions; menu dimensions are obsolete
- `Lua::Push`: now supports `std::unordered_map`
- `Maps::GetBiomeTypeRef` renamed to `Maps::getBiomeTypeRef` for consistency
- `Maps::GetBiomeType` renamed to `Maps::getBiomeType` for consistency
- **`Military`:**
  - New module for military functionality
  - new `makeSquad` to create a squad
  - changed `getSquadName` to take a squad identifier
  - new `updateRoomAssignments` for assigning a squad to a barracks and archery range
- **`Screen::Pen`:**
  - now accepts `top_of_text` and `bottom_of_text` properties to support offset text in graphics mode
  - now accepts `keep_lower` and `write_to_lower` properties to support foreground and background textures in graphics mode
- `Units::getCitizens()`: gets a list of citizens, which otherwise you'd have to iterate over all units the world to discover

Lua

- added two new window borders: `gui.BOLD_FRAME` for accented elements and `gui.INTERIOR_MEDIUM_FRAME` for a signature-less frame that's thicker than the existing `gui.INTERIOR_FRAME`
- Removed `os.execute()` and `io.popen()` built-in functions
- ***helpdb*:**
  - new function: `helpdb.refresh()` to force a refresh of the database. Call if you are a developer adding new scripts, loading new plugins, or changing help text during play
  - changed from auto-refreshing every 60 seconds to only refreshing on explicit call to `helpdb.refresh()`. docs very rarely change during a play session, and the automatic database refreshes were slowing down the startup of *gui/launcher* and anything else that displays help text.
- ***overlay*:**
  - overlay widgets can now specify focus paths for the viewscreens they attach to so they only appear in specific contexts. see *DFHack overlay dev guide* for details.
  - `OverlayWidget` now inherits from `Panel` instead of `Widget` to get all the frame and mouse integration goodies
- *tiletypes*: now has a Lua API! `tiletypes_setTile`
- `dfhack.gui.getDFViewscreen()`: returns the topmost underlying DF viewscreen
- `dfhack.gui.getSelectedCivZone`: returns the Zone that the user has selected currently
- `dfhack.job.attachJobItem()`: allows you to attach specific items to a job
- `dfhack.screen.paintTile()`: you can now explicitly clear the interface cursor from a map tile by passing `0` as the tile value
- `dfhack.units.getCitizens()`: gets a list of citizens

- `gui.CLEAR_PEN`: now clears the background and foreground and writes to the background (before it would always write to the foreground)
- `gui.KEEP_LOWER_PEN`: a general use pen that writes associated tiles to the foreground while keeping the existing background
- **`gui.View`:**
  - `visible` and `active` can now be functions that return a boolean
  - new function `view:getMouseFramePos()` for detecting whether the mouse is within (or over) the exterior frame of a view
- `gui.ZScreen`: Screen subclass that implements window raising, multi-viewscreen input handling, and viewscreen event pass-through so the underlying map can be interacted with and dragged around while DFHack screens are visible
- `maps.getBiomeType`: exposed preexisting function to Lua
- **`widgets.CycleHotkeyLabel`:**
  - options that are bare integers will no longer be interpreted as the pen color in addition to being the label and value
  - option labels and pens can now be functions that return a label or pen
  - add `label_below` attribute for compact 2-line output
  - Added `key_back` optional parameter to cycle backwards.
  - now supports rendering option labels in the color of your choice
  - new functions `setOption()` and `getOptionPen()`
- **`widgets.FilteredList`:**
  - Added `edit_on_change` optional parameter to allow a custom callback on filter edit change.
  - Added `case_sensitive` optional paramter to determine if filtering is case sensitive.
- **`widgets.HotkeyLabel`:**
  - Added `setLabel` method to allow easily updating the label text without mangling the keyboard shortcut.
  - Added `setOnActivate` method to allow easily updating the `on_activate` callback.
- **`widgets.Label`:**
  - tokens can now specify a `htile` property to indicate the tile that should be shown when the Label is hovered over with the mouse
  - click handlers no longer get the label itself as the first param to the click handler
  - token `tile` properties can now be functions that return a value
  - `label.scroll()` now understands `home` and `end` keywords for scrolling to the top or bottom
  - token `tile` properties can now be either pens or numeric texture ids
  - tiles can now have an associated width
- `widgets.List`: new callbacks for double click and shift double click
- `widgets.Panel`: new attributes to control window dragging and resizing with mouse or keyboard
- `widgets.TabBar`: new library widget (migrated from control-panel.lua)
- `widgets.ToggleHotkeyLabel`: now renders the `On` option in green text

- `widgets.Window`: Panel subclass with attributes preset for top-level windows

Structures

- add "hospital" language name category
- added missing tiletypes and corrected a few old ones based on a list supplied by Toady
- correct bit size of tree body data
- corrected alignment in `world.status`
- corrected misalignment in `historical_entity`
- corrected misalignment in `unitst` (affecting `occupation` and `adjective`)
- identified fields for squads and other military structures
- identified fields in `deep_vein_hollow`, `glowing_barrier`, and `cursed_tomb` map events
- identified some anons in `unitst` related to textures (thanks, putnam)
- identified two fields related to saves/autosaves to facilitate quicksave implementation
- identified two old and one new language name groups
- identified `divine_treasure` and `encased_horror` map events
- identify a table of daily events scheduled to take place inthe current year
- identify item vmethod 213 (applies a thread improvements to appropriate items based on an RNG)
- identify two anons in `difficultyst`
- identify various data types related to job completion/cancellation
- partially identified squad-related structures in `plotinfo` and corrected position of `civ_alert_idx` (does not affect alignment)
- realigned and fleshed out `entity_site_link` (again, thanks, putnam)
- realigned `furniture_type` enum (added BAG)
- realigned `stockpile_settings` for new "corpses" vector
- remove some no-longer-valid reputation types
- **Renamed globals to match DF:**
  - `ui`: renamed to `plotinfo`
  - `ui_advmode`: renamed to `adventure`
  - `ui_build_selector`: renamed to `buildreq`
  - `ui_sidebar_menus`: renamed to `game`
- split `gamest` into `gamest` and `gamest_extra` to accommodate steam-specific data in `gamest.mod_manager`
- `activity_info`: `unit_actor`, `unit_noble`, and `place` converted from pointers to integer references.
- `building_civzonest`: identify two variables, `dir_x` and `dir_y`, that handle archery range direction.
- `building_design`: corrected misalignments
- `creature_raw_graphics`: corrected misalignments
- `dipscript_popup`: `meeting_holder` converted from unit pointer into two unit refs `meeting_holder_actor` and `meeting_holder_noble`.

- `history_eventst`: Removed `history_event_masterpiece_created_arch_designst` and related enum value
- `item.setSharpness()`: more info about params
- `occupation_type`: add enum values for new occupations related to hospitals
- `plotinfost.``equipment``: Converted `items_unmanifested`, `items_unassigned`, and `items_assigned` vectors from pointers to item refs
- `plot_infost.``unk_8``: renamed to `theft_intrigues`. Fields `unk_1` thru `unk_8` renamed to `target_item`, `mastermind_hf`, `mastermind_plot_id`, `corruptor_hf`, `corruptor`, `corruptee_hf`, `corruptee`, and `theft_agreement`. `unk_1` renamed to `item_known_pos`.
- `specific_ref_type`: Removed `BUILDING_PARTY`, `PETINFO_PET`, and `PETINFO_OWNER` enum values to fix alignment.

Documentation

- added DFHack architecture diagrams to the dev intro
- added DFHack Quickstart guide
- the `untested` tag has been renamed to `unavailable` to better reflect the status of the remaining unavaialable tools. most of the simply "untested" tools have now been tested and marked as working. the remaining tools are known to need development work before they are available again.
- *Compilation*: instructions added for cross-compiling DFHack for Windows from a Linux Docker builder
- *devel/hello-world*: updated to be a better example from which to start new gui scripts
- *Installing*: updated to include Steam installation instructions
- *DFHack modding guide*: guide updated to include information for 3rd party script developers
- *DFHack overlay dev guide*: added troubleshooting tips and common development workflows

8.4.26 DFHack 0.47.05-r8

New Plugins

- *channel-safely*: auto-manage channel designations to keep dwarves safe
- *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
  - new action timer API for speeding up of slowing down units
- `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

8.4.27 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.

8.4.28 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 *deterioratecloth*es, *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
- ***The "Autostart" subtab*:**
  - 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 blueprint creation guide*:**
  - 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

- *deterioratecloths*: 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`

8.4.29 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`
- ***The "Autostart" subtab***:
  - 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)`

8.4.30 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*, *The "Autostart" subtab*: 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`

8.4.31 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

- *The "Autostart" subtab*: documentation for all of *Dreamfort*'s supporting files (useful for all forts, not just Dreamfort!)
- *Quickfort blueprint library*: updated dreamfort documentation and added screenshots

8.4.32 DFHack 0.47.05-r2

New Scripts

- *clear-webs*: removes all webs on the map and/or frees any webbed creatures
- *devel/block-borders*: overlay that displays map block borders
- *devel/luacov*: generate code test coverage reports for script development. Define the `DFHACK_ENABLE_LUACOV=1` environment variable to start gathering coverage metrics.
- *fix/drop-webs*: causes floating webs to fall to the ground
- *gui/blueprint*: interactive frontend for the *blueprint* plugin (with mouse support!)
- *gui/mass-remove*: mass removal/suspension tool for buildings and constructions
- *reveal-hidden-sites*: exposes all undiscovered sites
- *set-timeskip-duration*: changes the duration of the "Updating World" process preceding the start of a new game, enabling you to jump in earlier or later than usual

Fixes

- Fixed an issue preventing some external scripts from creating zones and other abstract buildings (see note about room definitions under "Internals")
- Fixed an issue where scrollable text in Lua-based screens could prevent other widgets from scrolling
- ***bodyswap*:**
  - stopped prior party members from tagging along after bodyswapping and reloading the map
  - made companions of bodyswapping targets get added to the adventurer party - they can now be viewed using the in-game party system
- ***buildingplan*:**
  - fixed an issue where planned constructions designated with DF's sizing keys (`umkh`) would sometimes be larger than requested
  - fixed an issue preventing other plugins like *automaterial* from planning constructions if the "enable all" buildingplan setting was turned on
  - made navigation keys work properly in the materials selection screen when alternate keybindings are used
- *color-schemes*: fixed an error in the `register` subcommand when the DF path contains certain punctuation characters
- *command-prompt*: fixed issues where overlays created by running certain commands (e.g. *gui/liquids*, *gui/teleport*) would not update the parent screen correctly
- *dwarfvet*: fixed a crash that could occur with hospitals overlapping with other buildings in certain ways

- *embark-assistant*: fixed faulty early exit in first search attempt when searching for waterfalls
- *gui/advfort*: fixed an issue where starting a workshop job while not standing at the center of the workshop required advancing time manually
- *gui/unit-info-viewer*: fixed size description displaying unrelated values instead of size
- *orders*: fixed crash when importing orders with malformed IDs
- ***quickfort***:
  - comments in blueprint cells no longer prevent the rest of the row from being read. A cell with a single '#' marker in it, though, will still stop the parser from reading further in the row.
  - fixed an off-by-one line number accounting in blueprints with implicit `#dig` modelines
  - changed to properly detect and report an error on sub-alias params with no values instead of just failing to apply the alias later (if you really want an empty value, use `{Empty}` instead)
  - improved handling of non-rectangular and non-solid extent-based structures (like fancy-shaped stockpiles and farm plots)
  - fixed conversion of numbers to DF keycodes in `#query` blueprints
  - fixed various errors with cropping across the map edge
  - properly reset config to default values in `quickfort reset` even if if the `dfhack-config/quickfort/quickfort.txt` config file doesn't mention all config vars. Also now works even if the config file doesn't exist.
- *stonesense*: fixed a crash that could occur when ctrl+scrolling or closing the Stonesense window
- `quickfortress.csv` blueprint: fixed refuse stockpile config and prevented stockpiles from covering stairways

Misc Improvements

- Added adjectives to item selection dialogs, used in tools like *gui/create-item* - this makes it possible to differentiate between different types of high/low boots, shields, etc. (some of which are procedurally generated)
- ***blueprint***:
  - made `depth` and `name` parameters optional. `depth` now defaults to `1` (current level only) and `name` defaults to "blueprint"
  - `depth` can now be negative, which will result in the blueprints being written from the highest z-level to the lowest. Before, blueprints were always written from the lowest z-level to the highest.
  - added the `--cursor` option to set the starting coordinate for the generated blueprints. A game cursor is no longer necessary if this option is used.
- *devel/annc-monitor*: added `report enable|disable` subcommand to filter combat reports
- *embark-assistant*: slightly improved performance of surveying and improved code a little
- *gui/advfort*: added workshop name to workshop UI
- ***quickfort***:
  - the Dreamfort blueprint set can now be comfortably built in a 1x1 embark
  - added the `--cursor` option for running a blueprint at specific coordinates instead of starting at the game cursor position
  - added more helpful error messages for invalid modeline markers
  - added support for extra space characters in blueprints

  - added a warning when an invalid alias is encountered instead of silently ignoring it
  - made more quiet when the `--quiet` parameter is specified
- *setfps*: improved error handling
- *stonesense*: sped up startup time
- *tweak* hide-priority: changed so that priorities stay hidden (or visible) when exiting and re-entering the designations menu
- *unretire-anyone*: the historical figure selection list now includes the `SYN_NAME` (necromancer, vampire, etc) of figures where applicable

API

- Added `dfhack.maps.getPlantAtTile(x, y, z)` and `dfhack.maps.getPlantAtTile(pos)`, and updated `dfhack.gui.getSelectedPlant()` to use it
- Added `dfhack.units.teleport(unit, pos)`

Internals

- Room definitions and extents are now created for abstract buildings so callers don't have to initialize the room structure themselves
- The DFHack test harness is now much easier to use for iterative development. Configuration can now be specified on the commandline, there are more test filter options, and the test harness can now easily rerun tests that have been run before.
- The `test/main` command to invoke the test harness has been renamed to just `test`
- Unit tests can now use `delay_until(predicate_fn, timeout_frames)` to delay until a condition is met
- Unit tests must now match any output expected to be printed via `dfhack.printerr()`
- Unit tests now support fortress mode (allowing tests that require a fortress map to be loaded) - note that these tests are skipped by continuous integration for now, pending a suitable test fortress

Lua

- new library: `argparse` is a collection of commandline argument processing functions
- **new string utility functions:**
  - `string:wrap(width)` wraps a string at space-separated word boundaries
  - `string:trim()` removes whitespace characters from the beginning and end of the string
  - `string:split(delimiter, plain)` splits a string with the given delimiter and returns a table of substrings. if `plain` is specified and set to `true`, `delimiter` is interpreted as a literal string instead of as a pattern (the default)
- new utility function: `utils.normalizePath()`: normalizes directory slashes across platoforms to `/` and coaleses adjacent directory separators
- *reveal*: now exposes `unhideFlood(pos)` functionality to Lua
- *xlsxreader*: added Lua class wrappers for the xlsxreader plugin API
- **`argparse.processArgsGetopt()` (previously `utils.processArgsGetopt()`):**

  - now returns negative numbers (e.g. `-10`) in the list of positional parameters instead of treating it as an option string equivalent to `-1 -0`
  - now properly handles `--` like GNU `getopt` as a marker to treat all further parameters as non-options
  - now detects when required arguments to long-form options are missing
- `gui.dwarfmode`: new function: `enterSidebarMode(sidebar_mode, max_esc)` which uses keypresses to get into the specified sidebar mode from whatever the current screen is
- `gui.Painter`: fixed error when calling `viewport()` method

Structures

- Identified remaining rhythm beat enum values
- `ui_advmode.interactions`: identified some fields related to party members
- `ui_advmode_menu`: identified several enum items
- **`ui_advmode`:**
  - identified several fields
  - renamed `wait` to `rest_mode` and changed to an enum with correct values
- `viewscreen_legendsst.cur_page`: added missing `Books` enum item, which fixes some other values

Documentation

- Added more client library implementations to the *remote interface docs*

8.4.33 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*

8.4.34 DFHack 0.47.04-r5

New Scripts

- *gui/quickfort*: fast access to the quickfort interactive dialog
- *workorder-recheck*: resets the selected work order to the `Checking` state

Fixes

- ***embark-assistant*:**
  - fixed order of factors when calculating min temperature
  - improved performance of surveying
- ***quickfort*:**
  - fixed eventual crashes when creating zones
  - fixed library aliases for tallow and iron, copper, and steel weapons
  - zones are now created in the active state by default
  - solve rare crash when changing UI modes
- *search*: fixed crash when searching the `k` sidebar and navigating to another tile with certain keys, like < or >
- *seedwatch*: fixed an issue where the plugin would disable itself on map load
- *stockflow*: fixed `j` character being intercepted when naming stockpiles
- *stockpiles*: no longer outputs hotkey help text beneath *stockflow* hotkey help text

Misc Improvements

- Lua label widgets (used in all standard message boxes) are now scrollable with Up/Down/PgUp/PgDn keys
- *autofarm*: now fallows farms if all plants have reached the desired count
- ***buildingplan*:**
  - added ability to set global settings from the console, e.g. `buildingplan set boulders false`
  - added "enable all" option for buildingplan (so you don't have to enable all building types individually). This setting is not persisted (just like quickfort_mode is not persisted), but it can be set from onMapLoad.init
  - modified `Planning Mode` status in the UI to show whether the plugin is in quickfort mode, "enable all" mode, or whether just the building type is enabled.
- ***quickfort*:**
  - Dreamfort blueprint set improvements: added a streamlined checklist for all required dreamfort commands and gave names to stockpiles, levers, bridges, and zones
  - added aliases for bronze weapons and armor
  - added alias for tradeable crafts
  - new blueprint mode: `#ignore`, useful for scratch space or personal notes
  - implement `{Empty}` keycode for use in quickfort aliases; useful for defining blank-by-default alias values

  - more flexible commandline parsing allowing for more natural parameter ordering (e.g. where you used to have to write `quickfort list dreamfort -l` you can now write `quickfort list -l dreamfort`)
  - print out blueprint names that a `#meta` blueprint is applying so it's easier to understand what meta blueprints are doing
  - whitespace is now allowed between a marker name and the opening parenthesis in blueprint modelines. for example, `#dig start (5; 5)` is now valid (you used to be required to write `#dig start(5; 5)`)

Lua

- `dfhack.run_command()`: changed to interface directly with the console when possible, which allows interactive commands and commands that detect the console encoding to work properly
- `processArgsGetopt()` added to utils.lua, providing a callback interface for parameter parsing and getopt-like flexibility for parameter ordering and combination (see docs in `library/lua/utils.lua` and `library/lua/3rdparty/alt_getopt.lua` for details).

Structures

- `job`: identified `order_id` field

Documentation

- Added documentation for Lua's `dfhack.run_command()` and variants

8.4.35 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 blueprint creation guide*: alias syntax and alias standard library documentation for *quickfort* blueprints
- *Quickfort blueprint library*: overview of the quickfort blueprint library

8.4.36 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

8.4.37 DFHack 0.47.04-r2

New Scripts

- *animal-control*: helps manage the butchery and gelding of animals
- *devel/kill-hf*: kills a historical figure
- *geld*: gelds or ungelds animals
- *list-agreements*: lists all guildhall and temple agreements
- *list-waves*: displays migration wave information for citizens/units
- *ungeld*: ungelds animals (wrapper around *geld*)

New Tweaks

- *tweak* do-job-now: adds a job priority toggle to the jobs list
- *tweak* reaction-gloves: adds an option to make reactions produce gloves in sets with correct handedness

Fixes

- Fixed a segfault when attempting to start a headless session with a graphical PRINT_MODE setting
- Fixed an issue with the macOS launcher failing to un-quarantine some files
- Fixed `Units::isEggLayer`, `Units::isGrazer`, `Units::isMilkable`, `Units::isTrainableHunting`, `Units::isTrainableWar`, and `Units::isTamable` ignoring the unit's caste
- Linux: fixed `dfhack.getDFPath()` (Lua) and `Process::getPath()` (C++) to always return the DF root path, even if the working directory has changed
- ***digfort*:**
  - fixed y-line tracking when .csv files contain lines with only commas

  - fixed an issue causing blueprints touching the southern or eastern edges of the map to be rejected (northern and western edges were already allowed). This allows blueprints that span the entire embark area.

- *embark-assistant*: fixed a couple of incursion handling bugs.
- *embark-skills*: fixed an issue with structures causing the `points` option to do nothing
- ***exportlegends*:**
  - fixed an issue where two different <reason> tags could be included in a <historical_event>
  - stopped including some tags with -1 values which don't provide useful information
- *getplants*: fixed issues causing plants to be collected even if they have no growths (or unripe growths)
- *gui/advfort*: fixed "operate pump" job
- *gui/load-screen*: fixed an issue causing longer timezones to be cut off
- ***labormanager*:**
  - fixed handling of new jobs in 0.47
  - fixed an issue preventing custom furnaces from being built
- ***modtools/moddable-gods*:**
  - fixed an error when creating the historical figure
  - removed unused `-domain` and `-description` arguments
  - made `-depictedAs` argument work
- ***names*:**
  - fixed an error preventing the script from working
  - fixed an issue causing renamed units to display their old name in legends mode and some other places
- *pref-adjust*: fixed some compatibility issues and a potential crash
- ***RemoteFortressReader*:**
  - fixed a couple crashes that could result from decoding invalid enum items (`site_realization_building_type` and `improvement_type`)
  - fixed an issue that could cause block coordinates to be incorrect
- *rendermax*: fixed a hang that could occur when enabling some renderers, notably on Linux
- ***stonesense*:**
  - fixed a crash when launching Stonesense
  - fixed some issues that could cause the splash screen to hang

Misc Improvements

- Linux/macOS: Added console keybindings for deleting words (Alt+Backspace and Alt+d in most terminals)
- ***add-recipe*:**
  - added tool recipes (minecarts, wheelbarrows, stepladders, etc.)
  - added a command explanation or error message when entering an invalid command
- *armoks-blessing*: added adjustments to values and needs
- ***blueprint*:**
  - now writes blueprints to the `blueprints/` subfolder instead of the df root folder
  - now automatically creates folder trees when organizing blueprints into subfolders (e.g. `blueprint 30 30 1 rooms/dining dig` will create the file `blueprints/rooms/dining-dig.csv`); previously it would fail if the `blueprints/rooms/` directory didn't already exist
- *confirm*: added a confirmation dialog for convicting dwarves of crimes
- *devel/query*: added many new query options
- ***digfort*:**
  - handled double quotes (") at the start of a string, allowing .csv files exported from spreadsheets to work without manual modification
  - documented that removing ramps, cutting trees, and gathering plants are indeed supported
  - added a `force` option to truncate blueprints if the full blueprint would extend off the edge of the map
- ***dwarf-op*:**
  - added ability to select dwarves based on migration wave
  - added ability to protect dwarves based on symbols in their custom professions
- ***exportlegends*:**
  - changed some flags to be represented by self-closing tags instead of true/false strings (e.g. `<is_volcano/>`) - note that this may require changes to other XML-parsing utilities
  - changed some enum values from numbers to their string representations
  - added ability to save all files to a subfolder, named after the region folder and date by default
- *gui/advfort*: added support for specifying the entity used to determine available resources
- *gui/gm-editor*: added support for automatically following ref-targets when pressing the `i` key
- *manipulator*: added a new column option to display units' goals
- *modtools/moddable-gods*: added support for `neuter` gender
- ***pref-adjust*:**
  - added support for adjusting just the selected dwarf
  - added a new `goth` profile
- *remove-stress*: added a `-value` argument to enable setting stress level directly
- *workorder*: changed default frequency from "Daily" to "OneTime"

API

- Added `Filesystem::mkdir_recursive`
- Extended `Filesystem::listdir_recursive` to optionally make returned filenames relative to the start directory
- `Units`: added goal-related functions: `getGoalType()`, `getGoalName()`, `isGoalAchieved()`

Internals

- Added support for splitting scripts into multiple files in the `scripts/internal` folder without polluting the output of *ls*

Lua

- Added a `ref_target` field to primitive field references, corresponding to the `ref-target` XML attribute
- Made `dfhack.units.getRaceNameById()`, `dfhack.units.getRaceBabyNameById()`, and `dfhack.units.getRaceChildNameById()` available to Lua

Ruby

- Updated `item_find` and `building_find` to use centralized logic that works on more screens

Structures

- Added a new `<df-other-vectors-type>`, which allows `world.*.other` collections of vectors to use the correct subtypes for items
- `creature_raw`: renamed `gender` to `sex` to match the field in `unit`, which is more frequently used
- `crime`: identified `witnesses`, which contains the data held by the old field named `reports`
- `intrigue`: new type (split out from `historical_figure_relationships`)
- `items_other_id`: removed `BAD`, and by extension, `world.items.other.BAD`, which was overlapping with `world.items.bad`
- `job_type`: added job types new to 0.47
- `plant_raw`: material_defs now contains arrays rather than loose fields
- `pronoun_type`: new enum (previously documented in field comments)
- `setup_character_info`: fixed a couple alignment issues (needed by *embark-skills*)
- `ui_advmode_menu`: identified some new enum items

Documentation

- Added some new dev-facing pages, including dedicated pages about the remote API, memory research, and documentation
- Expanded the installation guide
- Made a couple theme adjustments

8.4.38 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

8.4.39 DFHack 0.44.12-r3

New Plugins

- *autoclothing*: automatically manage clothing work orders
- *autofarm*: replaces the previous Ruby script of the same name, with some fixes
- *map-render*: allows programmatically rendering sections of the map that are off-screen
- *tailor*: automatically manages keeping your dorfs clothed

New Scripts

- *assign-attributes*: changes the attributes of a unit
- *assign-beliefs*: changes the beliefs of a unit
- *assign-facets*: changes the facets (traits) of a unit
- *assign-goals*: changes the goals of a unit
- *assign-preferences*: changes the preferences of a unit
- *assign-profile*: sets a dwarf's characteristics according to a predefined profile
- *assign-skills*: changes the skills of a unit
- *combat-harden*: sets a unit's combat-hardened value to a given percent
- *deep-embark*: allows embarking underground
- *devel/find-twbt*: finds a TWBT-related offset needed by the new *map-render* plugin
- *dwarf-op*: optimizes dwarves for fort-mode work; makes managing labors easier
- *forget-dead-body*: removes emotions associated with seeing a dead body
- *gui/create-tree*: creates a tree at the selected tile
- *linger*: takes over your killer in adventure mode
- *modtools/create-tree*: creates a tree
- *modtools/pref-edit*: add, remove, or edit the preferences of a unit

- *modtools/set-belief*: changes the beliefs (values) of units
- *modtools/set-need*: sets and edits unit needs
- *modtools/set-personality*: changes the personality of units
- *modtools/spawn-liquid*: spawns water or lava at the specified coordinates
- *set-orientation*: edits a unit's orientation
- *unretire-anyone*: turns any historical figure into a playable adventurer

Fixes

- Fixed a crash in the macOS/Linux console when the prompt was wider than the screen width
- Fixed inconsistent results from `Units::isGay` for asexual units
- Fixed some cases where Lua filtered lists would not properly intercept keys, potentially triggering other actions on the same screen
- ***autofarm*:**
  - fixed biome detection to properly determine crop assignments on surface farms
  - reimplemented as a C++ plugin to make proper biome detection possible
- *bodyswap*: fixed companion list not being updated often enough
- *cxxrandom*: removed some extraneous debug information
- *digfort*: now accounts for z-level changes when calculating maximum y dimension
- ***embark-assistant*:**
  - fixed bug causing crash on worlds without generated metals (as well as pruning vectors as originally intended).
  - fixed bug causing mineral matching to fail to cut off at the magma sea, reporting presence of things that aren't (like DF does currently).
  - fixed bug causing half of the river tiles not to be recognized.
  - added logic to detect some river tiles DF doesn't generate data for (but are definitely present).
- *eventful*: fixed invalid building ID in some building events
- *exportlegends*: now escapes special characters in names properly
- *getplants*: fixed designation of plants out of season (note that picked plants are still designated incorrectly)
- *gui/autogems*: fixed error when no world is loaded
- ***gui/companion-order*:**
  - fixed error when resetting group leaders
  - `leave` now properly removes companion links
- *gui/create-item*: fixed module support - can now be used from other scripts
- ***gui/stamper*:**
  - stopped "invert" from resetting the designation type
  - switched to using DF's designation keybindings instead of custom bindings
  - fixed some typos and text overlapping

- ***modtools/create-unit*:**
  - fixed an error associating historical entities with units
  - stopped recalculating health to avoid newly-created citizens triggering a "recover wounded" job
  - fixed units created in arena mode having blank names
  - fixed units created in arena mode having the wrong race and/or interaction effects applied after creating units manually in-game
  - stopped units from spawning with extra items or skills previously selected in the arena
  - stopped setting some unneeded flags that could result in glowing creature tiles
  - set units created in adventure mode to have no family, instead of being related to the first creature in the world
- ***modtools/reaction-product-trigger*:**
  - fixed an error dealing with reactions in adventure mode
  - blocked `\\BUILDING_ID` for adventure mode reactions
  - fixed `-clear` to work without passing other unneeded arguments
- ***modtools/reaction-trigger*:**
  - fixed a bug when determining whether a command was run
  - fixed handling of `-resetPolicy`
- *mousequery*: fixed calculation of map dimensions, which was sometimes preventing scrolling the map with the mouse when TWBT was enabled
- *RemoteFortressReader*: fixed a crash when a unit's path has a length of 0
- *stonesense*: fixed crash due to wagons and other soul-less creatures
- *tame*: now sets the civ ID of tamed animals (fixes compatibility with *autobutcher*)
- *title-folder*: silenced error when `PRINT_MODE` is set to `TEXT`

Misc Improvements

- Added a note to *dfhack-run* when called with no arguments (which is usually unintentional)
- On macOS, the launcher now attempts to un-quarantine the rest of DFHack
- *bodyswap*: added arena mode support
- *combine-drinks*: added more default output, similar to *combine-plants*
- *createitem*: added a list of valid castes to the "invalid caste" error message, for convenience
- *devel/export-dt-ini*: added more size information needed by newer Dwarf Therapist versions
- *dwarfmonitor*: enabled widgets to access other scripts and plugins by switching to the core Lua context
- ***embark-assistant*:**
  - added an in-game option to activate on the embark screen
  - changed waterfall detection to look for level drop rather than just presence
  - changed matching to take incursions, i.e. parts of other biomes, into consideration when evaluating tiles. This allows for e.g. finding multiple biomes on single tile embarks.

  - changed overlay display to show when incursion surveying is incomplete
  - changed overlay display to show evil weather
  - added optional parameter "fileresult" for crude external harness automated match support
  - improved focus movement logic to go to only required world tiles, increasing speed of subsequent searches considerably
- *exportlegends*: added rivers to custom XML export
- *exterminate*: added support for a special `enemy` caste
- ***gui/gm-unit*:**
  - added support for editing:
  - added attribute editor
  - added orientation editor
  - added editor for bodies and body parts
  - added color editor
  - added belief editor
  - added personality editor
- *modtools/create-item*: documented already-existing `-quality` option
- ***modtools/create-unit*:**
  - added the ability to specify `\\LOCAL` for the fort group entity
  - now enables the default labours for adult units with CAN_LEARN.
  - now sets historical figure orientation.
  - improved speed of creating multiple units at once
  - made the script usable as a module (from other scripts)
- ***modtools/reaction-trigger*:**
  - added `-ignoreWorker`: ignores the worker when selecting the targets
  - changed the default behavior to skip inactive/dead units; added `-dontSkipInactive` to include creatures that are inactive
  - added `-range`: controls how far elligible targets can be from the workshop
  - syndromes now are applied before commands are run, not after
  - if both a command and a syndrome are given, the command only runs if the syndrome could be applied
- *mousequery*: made it more clear when features are enabled
- ***RemoteFortressReader*:**
  - added a basic framework for controlling and reading the menus in DF (currently only supports the building menu)
  - added support for reading item raws
  - added a check for whether or not the game is currently saving or loading, for utilities to check if it's safe to read from DF
  - added unit facing direction estimate and position within tiles

  - added unit age
  - added unit wounds
  - added tree information
  - added check for units' current jobs when calculating the direction they are facing

API

- Added new `plugin_load_data` and `plugin_save_data` events for plugins to load/save persistent data
- Added `Maps::GetBiomeType` and `Maps::GetBiomeTypeByRef` to infer biome types properly
- Added `Units::getPhysicalDescription` (note that this depends on the `unit_get_physical_description` offset, which is not yet available for all DF builds)

Internals

- Added new Persistence module
- Cut down on internal DFHack dependencies to improve build times
- Improved concurrency in event and server handlers
- Persistent data is now stored in JSON files instead of historical figures - existing data will be migrated when saving
- *stonesense*: fixed some OpenGL build issues on Linux

Lua

- Exposed `gui.dwarfmode.get_movement_delta` and `gui.dwarfmode.get_hotkey_target`
- `dfhack.run_command` now returns the command's return code

Ruby

- Made `unit_ishostile` consistently return a boolean

Structures

- Added `unit_get_physical_description` function offset on some platforms
- **Added/identified types:**
  - `assume_identity_mode`
  - `musical_form_purpose`
  - `musical_form_style`
  - `musical_form_pitch_style`
  - `musical_form_feature`
  - `musical_form_vocals`
  - `musical_form_melodies`

    - `musical_form_interval`
    - `unit_emotion_memory`
- `need_type`: fixed `PrayOrMeditate` typo
- `personality_facet_type`, `value_type`: added `NONE` values
- `twbt_render_map`: added for 64-bit 0.44.12 (for *map-render*)

8.4.40 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

8.4.41 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_civliststs`)
- `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_civliststs`:**
  - 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_reportliststs`: added new mission and spoils report-related fields (fixed layout)
- `world.languages`: identified (minimal information; whole languages stored elsewhere)
- **`world.status`:**
  - `mission_reports`: renamed, was `missions`
  - `spoils_reports`: identified
- `world.unk_131ec0`, `world.unk_131ef0`: researched layout
- `world.worldgen_status`: identified many fields
- `world`: `belief_systems`: identified

8.4.42 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)

8.4.43 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)

8.4.44 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`

8.4.45 DFHack 0.44.05-r2

New Plugins

- *embark-assistant*: adds more information and features to embark screen

New Scripts

- *adv-fix-sleepers*: fixes units in adventure mode who refuse to wake up (Bug 6798)
- *hermit*: blocks caravans, migrants, diplomats (for hermit challenge)

New Features

- With `PRINT_MODE:TEXT`, setting the `DFHACK_HEADLESS` environment variable will hide DF's display and allow the console to be used normally. (Note that this is intended for testing and is not very useful for actual gameplay.)

Fixes

- *devel/export-dt-ini*: fix language_name offsets for DT 39.2+
- *devel/inject-raws*: fixed gloves and shoes (old typo causing errors)
- *RemoteFortressReader*: fixed an issue with not all engravings being included
- *view-item-info*: fixed an error with some shields

Misc Improvements

- *adv-rumors*: added more keywords, including names
- *autochop*: can now exclude trees that produce fruit, food, or cookable items
- *RemoteFortressReader*: added plant type support

8.4.46 DFHack 0.44.05-r1

New Scripts

- *break-dance*: Breaks up a stuck dance activity
- *devel/check-other-ids*: Checks the validity of "other" vectors in the `world` global
- *devel/dump-offsets*: prints an XML version of the global table included in in DF
- *fillneeds*: Use with a unit selected to make them focused and unstressed
- *firestarter*: Lights things on fire: items, locations, entire inventories even!
- *flashstep*: Teleports adventurer to cursor
- *ghostly*: Turns an adventurer into a ghost or back
- *gui/cp437-table*: An in-game CP437 table
- *questport*: Sends your adventurer to the location of your quest log cursor

- *view-unit-reports*: opens the reports screen with combat reports for the selected unit

Fixes

- Fixed a crash that could occur if a symbol table in symbols.xml had no content
- Fixed issues with the console output color affecting the prompt on Windows
- *autolabor*, *autohauler*, *labormanager*: added support for "put item on display" jobs and building/destroying display furniture
- *createitem*: stopped items from teleporting away in some forts
- ***devel/inject-raws*:**
  - now recognizes spaces in reaction names
  - now recognizes spaces in reaction names
- *dig*: added support for designation priorities - fixes issues with designations from `digv` and related commands having extremely high priority
- ***dwarfmonitor*:**
  - fixed display of creatures and poetic/music/dance forms on `prefs` screen
  - added "view unit" option
  - now exposes the selected unit to other tools
- *exportlegends*: fixed an error that could occur when exporting empty lists
- *gui/gm-editor*: fixed an error when editing primitives in Lua tables
- *gui/gm-unit*: can now edit mining skill
- *gui/quickcmd*: stopped error from adding too many commands
- *modtools/create-unit*: fixed error when domesticating units
- *names*: fixed many errors
- *quicksave*: fixed an issue where the "Saving…" indicator often wouldn't appear

Misc Improvements

- The console now provides suggestions for built-in commands
- *binpatch*: now reports errors for empty patch files
- *devel/export-dt-ini*: avoid hardcoding flags
- ***exportlegends*:**
  - reordered some tags to match DF's order
  - added progress indicators for exporting long lists
- *force*: now provides useful help
- ***full-heal*:**
  - can now select corpses to resurrect
  - now resets body part temperatures upon resurrection to prevent creatures from freezing/melting again

    - now resets units' vanish countdown to reverse effects of *exterminate*
- *gui/gm-editor*: added enum names to enum edit dialogs
- ***gui/gm-unit*:**
    - added a profession editor
    - misc. layout improvements
    - made skill search case-insensitive
- *gui/liquids*: added more keybindings: 0-7 to change liquid level, P/B to cycle backwards
- *gui/pathable*: added tile types to sidebar
- *gui/rename*: added "clear" and "special characters" options
- *launch*: can now ride creatures
- ***modtools/skill-change*:**
    - now updates skill levels appropriately
    - only prints output if `-loud` is passed
- *names*: can now edit names of units
- ***RemoteFortressReader*:**
    - support for moving adventurers
    - support for vehicles, gem shapes, item volume, art images, item improvements
    - includes item stack sizes
    - some performance improvements

Removed

- *tweak*: `kitchen-keys`: Bug 614 fixed in DF 0.44.04
- *warn-stuck-trees*: Bug 9252 fixed in DF 0.44.01

Internals

- `Gui::getAnyUnit()` supports many more screens/menus

Lua

- Added a new `dfhack.console` API
- API can now wrap functions with 12 or 13 parameters
- Exposed `get_vector()` (from C++) for all types that support `find()`, e.g. `df.unit.get_vector() == df.global.world.units.all`
- Improved `json` I/O error messages
- Stopped a crash when trying to create instances of classes whose vtable addresses are not available

Structures

- Added `buildings_other_id.DISPLAY_CASE`
- Added `job_type.PutItemOnDisplay`
- Added `twbt_render_map` code offset on x64
- Fixed an issue preventing `enabler` from being allocated by DFHack
- Fixed `unit` alignment
- Fixed `viewscreen_titlest.start_savegames` alignment
- Found `renderer` vtable on osx64
- Identified `historical_entity.unknown1b.deities` (deity IDs)
- Located `start_dwarf_count` offset for all builds except 64-bit Linux; *startdwarf* should work now
- **New globals:**
  - `soul_next_id`
  - `version`
  - `min_load_version`
  - `movie_version`
  - `basic_seed`
  - `title`
  - `title_spaced`
  - `ui_building_resize_radius`
- The former `announcements` global is now a field in `d_init`
- The `ui_menu_width` global is now a 2-byte array; the second item is the former `ui_area_map_width` global, which is now removed
- `adventure_movement_optionst`, `adventure_movement_hold_tilest`, `adventure_movement_climbst`: named coordinate fields
- `artifact_record`: fixed layout (changed in 0.44.04)
- `incident`: fixed layout (changed in 0.44.01) - note that many fields have moved
- `mission`: added type
- `unit`: added 3 new vmethods: `getCreatureTile`, `getCorpseTile`, `getGlowTile`
- `viewscreen_assign_display_itemst`: fixed layout on x64 and identified many fields
- `viewscreen_reportliststst`: fixed layout, added `mission_id` vector
- `world.status`: named `missions` vector
- `world` fields formerly beginning with `job_` are now fields of `world.jobs`, e.g. `world.job_list` is now `world.jobs.list`

8.4.47 Older Changelogs

Are kept in a separate file: History

INDEX OF DFHACK TOOLS

d

e

f

g

h

i

k

l

m

n

t

u

w

x

"ADVENTURE" TAG INDEX - TOOLS THAT ARE USEFUL WHILE IN ADVENTURE MODE.

"DFHACK" TAG INDEX - TOOLS THAT YOU USE TO RUN DFHACK COMMANDS OR INTERACT WITH THE DFHACK OR DF SYSTEM.

S

t

u

"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.

d

e

f

g

h

i

l

m

n

o

p

q

r

s

t

u

w

"LEGENDS" TAG INDEX - TOOLS THAT ARE USEFUL WHILE IN LEGENDS MODE.

"ARMOK" TAG INDEX - TOOLS WHICH GIVE THE PLAYER GOD-LIKE POWERS OR THE ABILITY TO ACCESS INFORMATION THE GAME INTENTIONALLY KEEPS HIDDEN. PLAYERS THAT DO NOT WISH TO SEE THESE TOOLS CAN HIDE THEM IN THE “PREFERENCES“ TAB OF ‘GUI/CONTROL-PANEL‘.

h

l

m

p

r

s

t

u

w

"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 WITH FORT LAYOUT.

"DEV" TAG INDEX - TOOLS THAT ARE USEFUL WHEN DEBUGGING OR DEVELOPING MODS.

p

r

x

"FPS" TAG INDEX - TOOLS THAT HELP YOU PREVENT IMPACT TO YOUR FPS.

"GAMEPLAY" TAG INDEX - TOOLS THAT INTRODUCE NEW GAMEPLAY ELEMENTS.

"INSPECTION" TAG INDEX - TOOLS THAT LET YOU VIEW INFORMATION THAT IS OTHERWISE DIFFICULT TO FIND.

S

t

u

w

"PRODUCTIVITY" TAG INDEX - TOOLS THAT HELP YOU PERFORM COMMON TASKS QUICKLY AND EASILY.

U

W

"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.

l

r

s

"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.

l

p

q

r

s

t

u

w

"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.

l

m

p

r

s

t

w

"WORKORDERS" TAG INDEX - TOOLS THAT INTERACT WITH WORKORDERS.

"UNAVAILABLE" TAG INDEX - TOOLS THAT ARE NOT YET AVAILABLE FOR THE CURRENT RELEASE.

n

p

q

r

s

t

v

w

z