MacPorts Guide

Mark Duling

Dr. Michael A Maibaum

Will Barton

Chapter 1. Introduction

MacPorts is an easy to use system for compiling, installing, and managing open source software. MacPorts may be conceptually divided into two main parts: the infrastructure, known as MacPorts base, and the set of available ports. A MacPorts port is a set of specifications contained in a Portfile that defines an application, its characteristics, and any files or special instructions required to install it. This allows you to use a single command to tell MacPorts to automatically download, compile, and install applications and libraries. But using MacPorts to manage your open source software provides several other significant advantages. For example, MacPorts:

- Installs automatically any required support software, known as dependencies, for a given port.
- Provides for uninstalls and upgrades for installed ports.
- Confines ported software to a private "sandbox" that keeps it from intermingling with your operating system and its vendor-supplied software to prevent them from becoming corrupted.
- Allows you to create pre-compiled binary installers of ported applications to quickly install software on remote computers without compiling from source code.

MacPorts is developed on Mac OS X, though it is designed to be portable so it can work on other Unix-like systems, especially those descended from the Berkeley Software Distribution (BSD).

The following notational conventions are used in the MacPorts Guide to distinguish between terminal input/output, file text, and other special text types.

- Terminal I/O and file text.

```
%% Commands to be typed into a terminal window.
```

```
Command output to a terminal window.
```

```
File text.
```

- Other special text types.

  A hyperlink: spontaneous combustion.

  A file: `/var/log/system.log`.

  A command: **ifconfig**.

  An option: port `install`

Chapter 2. Installing MacPorts

This chapter shows you how to install MacPorts and its prerequisites step-by-step. Note that the sections about installing X11 and installing Xcode are Mac OS X specific. If you wish to install MacPorts on another platform, first make sure you have X11 and gcc installed, and then skip ahead to installing MacPorts from source and continue to the end of the chapter.

2.1. Install X11

It is recommended that you install the X Window System (X11) even if you don't plan to run X11 applications immediately. Apple's X11 server is normally used to display X11 applications, though you could also install the xorg-server port. If Apple's X11 wasn't installed when Mac OS X was installed, follow these steps.

1. Insert the "Mac OS X Install Disk" and run the package named "Optional Installs".
2. At the software selection window expand the **Applications** category and click the check box beside **X11** (and

nothing else).

3. Click Install to install X11.

Before launching an X11 application on Mac OS X 10.4, you must open X11.app and start an xterm session. Later OS versions should launch X11.app automatically when an X11 application is run from the Terminal.

```
%% xterm
```

After the X11 session window opens, you may launch X11 apps from another terminal window. See Optional X11 Settings if you wish to launch X11 applications from an X11 session window.

> **Note**
>
> X11 and the X11SDK (from Xcode Tools) are both required for X11 apps. To verify the presence of both, check for files `com.apple.pkg.X11User.bom` & `com.apple.pkg.X11SDKLeo.bom` in `/Library/Receipts/boms/`. On Mac OS X 10.4, look for files `X11User.pkg` & `X11SDK.pkg` in `/Library/Receipts/`.

2.1.1. Optional X11 Settings

To launch X11 applications directly from an X11 window (instead of a regular terminal window), you need to make it so X11 sessions opened using the menu bar respect your `.profile` file.

1. Open X11 and select Customize Menu ... from the **Applications** menu.
2. Double-click the menu item Terminal and change: "xterm" to "xterm -ls"
3. Click Done to save the change.

2.2. Install Xcode

To install Xcode and the X11 SDK, follow the steps below.

> **Note**
>
> Always make sure to install the latest available version of Xcode for your Mac OS X release; using outdated versions of Xcode may cause port install failures. Also note that Xcode is not updated via Mac OS X's Software Update utility on OS versions prior to 10.6.

1. Download the latest version of Xcode if you have the currently shipping release of Mac OS X.

   > **Note**
   >
   > Downloading Xcode from the Mac App Store does not install it. You need to manually run the installer in your Applications folder after downloading it.

   If you are using Mac OS X 10.6, there are two branches of Xcode which could be considered to be the latest, 3.2.x and 4.0.x. Xcode 4 costs money, but Xcode 3 is still available free of charge. There are two options for downloading it:

   a. Xcode 3.2.2 - smaller download, but you will need to run Software Update after installing to get the latest version.

   b. Xcode 3.2.6 and iOS SDK 4.3 - includes iOS SDK which is not needed for MacPorts.

   You may also be able to install Xcode 3.2 from your Mac OS X 10.6 DVD and then run Software Update to get the latest version.

   If you have an earlier release of Mac OS X, you may download the latest version of Xcode for OS X 10.5 (v3.1.4) or for 10.4 (v2.5).

2. Run the Xcode package installer.
3. Ensure that those of the following options that are available in the installer for your version of Xcode are selected:

   a. UNIX Development
   b. System Tools
   c. X11 SDK
   d. Command Line Support

4. Click Install to install Xcode.

2.3. Install MacPorts

If you are using Mac OS X, you should install MacPorts using the Mac OS X package installer unless you do not wish to install it to `/opt/local/`, the default MacPorts location, or if you wish to install a pre-release version of MacPorts base. However, if you wish to install multiple copies of MacPorts or install MacPorts on another OS platform, you must install MacPorts from the source code.

> **Note**
>
> Though a distinction is made between pre-release and release versions of MacPorts base, the ports collection supports no such distinction or versioning. The selfupdate command installs the latest port revisions from Subversion (at a slight delay), and updates MacPorts base to the latest released version.

2.3.1. Mac OS X Package Install

The Mac OS X package installer automatically installs MacPorts, sets the shell environment, and runs a selfupdate operation to update the ports tree and MacPorts base with the latest release.

1. Download the latest `MacPorts-x.x.x.dmg` disk image (whose name does not contain -beta or -rc) from the MacPorts download directory.
2. Double-click the `MacPorts-x.x.x.pkg` package installer on the disk image.
3. Perform the default "easy" install.

2.3.2. Source Install

If you installed MacPorts using the package installer, skip this section. To install MacPorts from the source code, follow the steps below.

1. Download and unzip the latest MacPorts tarball from the MacPorts download directory.
2. Perform the commands shown in a terminal window. If you wish to use a path other than `/opt/local`, use the option `--prefix` and substitute a path for NEW_PREFIX.

```
%% cd ~/MacPorts-x.x.x/
%% ./configure --prefix=NEW_PREFIX  (setting prefix is optional)
%% make
%% sudo make install
```

2.3.3. Subversion Install

If you installed MacPorts using the package installer, skip this section.

There are times when some may want to run MacPorts from a version newer than the current stable release. Maybe there's a new feature that you'd like to use, or it fixes an issue you've encountered, or you just like to be on the cutting edge. These steps explain how to run completely from trunk, using only Subversion to keep MacPorts up to date.

1. Check out MacPorts source

   Pick a location to store a working copy of the MacPorts code. For this example, `/opt/mports` will be used, but you can put the source anywhere. This example will create `/opt/mports/trunk` containing everything needed for MacPorts.

```
%% mkdir -p /opt/mports
%% cd /opt/mports
%% svn checkout https://svn.macports.org/repository/macports/trunk
```

   > **Note**
   >
   > You only really need the base subdirectory to run MacPorts, so you can avoid checking out the rest if you don't want to use a Subversion-based ports tree (see Step 3 below). To just get the base directory, append `/base` to the end of the svn checkout command above. The resulting directory will then be `/opt/mports/base`

2. Build and Install MacPorts

   MacPorts uses autoconf and makefiles for installation. These commands will build and install MacPorts to `/opt/local`. You can add the `--prefix` option to `./configure` to relocate MacPorts to another directory if needed.

```
%% cd /opt/mports/trunk/base
%% ./configure --enable-readline
%% make
%% sudo make install
%% make distclean
```

3. (Optional) Configure MacPorts to use port information from Subversion

   This step is useful if you want to do port development. Open `/opt/local/etc/macports/sources.conf` in a text editor. The last line which should look like this:

   ```
   rsync://rsync.macports.org/release/tarballs/ports.tar [default]
   ```

   Change it to point to the working copy you checked out:

   ```
   file:///opt/mports/trunk/dports [default]
   ```

   Now MacPorts will look for portfiles in the working copy.

4. Environment

   You should setup your PATH and other environment options according to the following section.

2.3.4. Install Multiple MacPorts Copies

Occasionally a MacPorts developer may wish to install more than one MacPorts instance on the same host. Only one copy of MacPorts may use the default prefix `/opt/local`, so for additional installations use the option `--prefix` as shown below. It's also recommended to change the applications dir using `--with-applications-dir` to avoid conflicts in `/Applications/MacPorts`.

> **Note**
>
> The first command temporarily removes the standard MacPorts binary paths because they must not be present while installing a second instance.

```
%% export PATH=/bin:/sbin:/usr/bin:/usr/sbin
%% MP_PREFIX=/opt/macports-test
%% ./configure --prefix=$MP_PREFIX --with-applications-dir=$MP_PREFIX/Applications
%% make
%% sudo make install
```

2.4. MacPorts Upgrade

MacPorts base upgrades are performed automatically (when a newer release is available) during a selfupdate operation. To upgrade a copy of MacPorts that was installed from source to the newer release of the source code, simply repeat the source install with the newer version of the MacPorts source code.

2.5. Uninstall

Uninstalling MacPorts can be a drastic step, and depending on the issue you are experiencing, you may not need to do so. If you are unsure, ask on the macports-users mailing list first.

If you need to uninstall MacPorts, and your **port** command is functioning, first uninstall all the installed ports by running this command in the Terminal:

```
%% sudo port -fp uninstall --follow-dependents installed
```

All that will be left in your installation prefix now will be files that were not registered to any port. This includes configuration files, databases, any files which MacPorts renamed in order to allow a forced installation or upgrade, and the base MacPorts software itself. You may wish to save your configuration files (most are in `${prefix}/etc`), databases, or any other unique data by moving it aside.

To remove all remaining traces of MacPorts, run the following command in the Terminal. If you have changed `prefix`, `applications_dir` or `frameworks_dir` from their default values, then replace `/opt/local` with your `prefix`, replace `/Applications/MacPorts` with your `applications_dir`, and/or add your `frameworks_dir` to the list, respectively.

```
%% sudo rm -rf \
    /opt/local \
    /Applications/DarwinPorts \
    /Applications/MacPorts \
    /Library/LaunchDaemons/org.macports.* \
    /Library/Receipts/DarwinPorts*.pkg \
    /Library/Receipts/MacPorts*.pkg \
    /Library/StartupItems/DarwinPortsStartup \
    /Library/Tcl/darwinports1.0 \
    /Library/Tcl/macports1.0 \
    ~/.macports
```

If you use a shell other than bash (perhaps tcsh), you may need to adjust the above to fit your shell's syntax. Also note that depending on which version of MacPorts you have and which ports you have installed, not all of the above paths will exist on your system. This is OK.

2.6. MacPorts and the Shell

MacPorts requires that some environment variables be set in the shell. When MacPorts is installed using the Mac OS X package installer, a "postflight" script is run after installation that automatically adds or modifies a shell configuration file in your home directory, ensuring that it defines variables according to the rules described in the following section. Those installing MacPorts from source code must modify their environment manually using the rules as a guide.

Depending on your shell and which configuration files already exist, the installer may use `.profile`, `.bash_login`, `.bash_profile`, `.tcshrc`, or `.cshrc`.

2.6.1. The Postflight Script

The postflight script automatically sets the `PATH` variable, and optionally the `MANPATH` and `DISPLAY` variables according to the rules described below. If a current shell configuration file exists at installation time it is renamed to "mpsaved_$timestamp". Those installing MacPorts from source code must modify their environment manually using the rules as a guide.

- Required: `PATH` variable

  This variable is set by the postflight script to append the MacPorts executable paths to the default path as shown. The MacPorts paths are appended at the front of `PATH` so the MacPorts libraries will take precedence over vendor-supplied libraries for ported software at runtime.

  ```
  export PATH=/opt/local/bin:/opt/local/sbin:$PATH
  ```

  Note

  The user environment's $PATH is not in effect while ports are being installed, because the $PATH is scrubbed before ports are installed, and restored afterwards. To change the search path for locating system executables (rsync, tar, etc.) during port installation, see the macports.conf file variable `binpath`. But changing this variable is for advanced users only, and is not generally needed or recomended.

- Optional: `MANPATH` variable

  Condition: If prior to MacPorts installation a `MANPATH` variable exists in a current `.profile` that contains neither the value `${prefix}/share/man,` nor any empty values, the postflight script sets the `MANPATH` variable as shown below. Otherwise, the `MANPATH` variable is omitted.

  ```
  export MANPATH=/opt/local/share/man:$MANPATH
  ```

  Here are some examples of paths that contain empty values:

  /usr/share/man:
  :/usr/share/man
  /usr/share/man::/usr/X11R6/man

- Optional: `DISPLAY` variable

  Condition: If installing on a Mac OS X version earlier than 10.5 (Leopard), and if a shell configuration file exists at time of MacPorts installation without a `DISPLAY` variable, the postflight script sets a `DISPLAY` variable as shown below. The `DISPLAY` variable is always omitted on Mac OS X 10.5 or higher.

  ```
  export DISPLAY=:0.0
  ```

2.6.2. Verify the configuration file

To verify that the file containing the MacPorts variables is in effect, type **env** in the terminal to verify the current environment settings after the file has been created. Example output for the **env** command is shown below.

Note

Changes to shell configuration files do not take effect until a new terminal session is opened.

```
MANPATH=
TERM_PROGRAM=Apple_Terminal
TERM=xterm-color
SHELL=/bin/bash
TERM_PROGRAM_VERSION=237
USER=joebob
__CF_USER_TEXT_ENCODING=0x1FC:0:0
PATH=/opt/local/bin:/opt/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin
PWD=/Users/joebob

EDITOR=/usr/bin/pico
```

```
EDITOR=/usr/bin/pico
SHLVL=1
HOME=/Users/joebob
LOGNAME=joebob
DISPLAY=:0.0
SECURITYSESSIONID=b0cea0
_=/usr/bin/env
```

2.6.3. Optional Editor Variables

You can set an environment variable in order to use your favorite text editor with edit option of port command.

MacPorts will check `MP_EDITOR`, `VISUAL` and `EDITOR` in this order, allowing you to either use a default editor shared with other programs (`VISUAL` and `EDITOR`) or a Macports' specific one (`MP_EDITOR`).

For example, to use the nano editor, add this line to your bash config:

```
export EDITOR=/usr/bin/nano
```

To use the user-friendly GUI editor TextWrangler (installation required), add this line:

```
export EDITOR=/usr/bin/edit
```

To keep a command-line text editor as default while using a graphic editor with portfiles, add this:

```
    export EDITOR=/usr/bin/vi
    export MP_EDITOR=/usr/bin/edit
```

Chapter 3. Using MacPorts

This chapter describes using the **port** command, port variants, common tasks and port binaries.

3.1. The port Command

The MacPorts **port** command is the main utility used to interact with MacPorts. It is used to update `Portfile`s and the MacPorts infrastructure, and install and manage ports.

3.1.1. help

The `help` action shows some brief information about the specified action, or if no action is specified, shows basic usage information for the **port** command in general.

```
%% port help selfupdate
```

```
Usage: selfupdate --nosync

Upgrade MacPorts itself and run the sync target
```

3.1.2. selfupdate

The `selfupdate` action should be used regularly to `sync` the local ports tree with the global MacPorts ports repository so you will have the latest port versions. It also checks for new revisions of the MacPorts infrastructure, called MacPorts base, and upgrades it when necessary.

Note

Selfupdate runs only on Mac OS X. If you are running MacPorts on another platform, you must use the sync action to update the ports tree; to update MacPorts base you must manually install a newer version from source.

```
%% sudo port selfupdate
```

Use the debug flag for verbose output.

```
%% sudo port -d selfupdate
```

```
DEBUG: Rebuilding the MacPorts base system if needed.
DEBUG: Synchronizing ports tree(s)
Synchronizing from rsync://rsync.macports.org/release/ports/
DEBUG: /usr/bin/rsync -rtzv --delete-after rsync://rsync.macports.org/release/ports/
receiving file list ... done

[ ... trimmed ... ]
```

```
The MacPorts installation is not outdated and so was not updated
DEBUG: Setting ownership to root
selfupdate done!
```

If selfupdate detects that a newer version of MacPorts base is available, it automatically updates the installed copy of MacPorts base to the latest released version. In that case, you will see the upgrade Makefile execute, and when it finishes you will see this message:

```
DEBUG: Updating using rsync
receiving file list ... done

Congratulations, you have successfully installed the MacPorts system.
```

3.1.3. sync

The `sync` action performs a subset of `selfupdate` actions. It synchronizes the ports tree, as does `selfupdate`, but it does not check for upgrades to MacPorts base. On Mac OS X, unless there is a special reason not to do so, you should run selfupdate.

> **Note**
>
> For platforms other than Mac OS X, `sync` is the only way to get port updates because `selfupdate` is supported only on Mac OS X.

3.1.4. list

The `list` action lists the currently available version of the specified ports, or if no ports are specified, displays a list of all available ports. The list of available ports is very long, so use search if you know a port's name or part of it.

```
%% port list
```

3.1.5. search

The `search` action allows finding ports by partial matches of the name or description. Other fields can be matched against, and matched in different ways, by using options. Run **`port help search`** for details.

```
%% port search rrd
```

```
php5-rrdtool @1.0 (php, net, devel)
    PHP 5 glue for rrdtool

rrdtool @1.4.4 (net)
    Round Robin Database

Found 2 ports.
```

3.1.6. info

The `info` action is used to get information about a port: description, maintainer, etc.

```
%% port info flowd
```

```
flowd @0.9 (net)
Variants:             universal

Description:          flowd is a small, fast and secure NetFlow collector.
Homepage:             http://www.mindrot.org/flowd.html

Platforms:            darwin
License:              unknown
Maintainers:          nomaintainer@macports.org
```

3.1.7. deps

The `deps` action shows you the dependencies of a port; dependencies are explicitly declared in `Portfile`s.

```
%% port deps apache2
```

```
Full Name: apache2 @2.2.17_0+preforkmpm
Library Dependencies: apr, apr-util, expat, openssl, pcre
```

3.1.8. variants

The `variants` action allows you to check what variations of a port are available before you install it. Variants are a way

for port authors to provide options that may be invoked at install time. See Invoking Port Variants below to install ports that have variants.

```
%% port variants nmap
```

```
nmap has the variants:
   no_pcre: build without pcre support
   no_ssl: build without ssl support
   universal: Build for multiple architectures
   zenmap: build zenmap in addition to nmap
```

3.1.9. install

The action `install` is used to install a port. See Invoking Port Variants below to install ports that have variants.

```
%% sudo port install nmap
```

Note

You may break up a port's installation into smaller steps for troubleshooting by passing **port** a prior installation phase such as fetch, configure, build, or destroot. See section Port Phases for a complete list of phases.

3.1.10. clean

The action `clean` deletes all intermediate files that MacPorts creates while building a port. A **`port clean`** is also often necessary to remove corrupted tarballs after a failed **`fetch`** phase.

```
%% sudo port clean --all vile
```

Note

You may also clean files selectively by using options `--dist`, `--archive`, or `--work`.

3.1.11. uninstall

The `uninstall` action will remove an installed port.

```
%% sudo port uninstall vile
```

Note

To also recursively uninstall the ports that the given port depends on, use the `--follow-dependencies` flag. This will not uninstall dependencies that are marked as requested or that have other dependents.

```
%% sudo port uninstall --follow-dependencies vile
```

To recursively uninstall all ports that depend on the given port before uninstalling the port itself, use the `--follow-dependents` flag.

```
%% sudo port uninstall --follow-dependents ncurses
```

If a port is a dependency of another installed port, uninstall will not remove it unless you remove the dependent port(s) first. To override this behavior, use the `-f` (force) switch. This will obviously break the dependents. Don't force uninstall ports unless you know what you are doing.

```
%% sudo port -f uninstall ncurses
```

3.1.12. contents

The `contents` action displays the files that have been installed by a given port. Uninstalled ports cannot have their contents listed.

```
%% port contents xorg-renderproto
```

```
Port xorg-renderproto contains:
  /opt/local/include/X11/extensions/render.h
  /opt/local/include/X11/extensions/renderproto.h
  /opt/local/lib/pkgconfig/renderproto.pc
  /opt/local/share/doc/renderproto/renderproto.txt
```

3.1.13. installed

The `installed` action displays the installed versions, variants and activation status of the specified ports, or if no ports

are specified, all installed ports. Use the -v option to also display the platform and CPU architecture(s) for which the ports were built, and any variants which were explicitly negated.

```
%% port installed
```

```
The following ports are currently installed:
  aalib @1.4rc5_2 (active)
  apr @1.2.8_0 (active)
  apr-util @1.2.8_2 (active)
  atk @1.18.0_0 (active)

[ ... trimmed ...]

  wxWidgets @2.8.4_2 (active)
  Xft2 @2.1.7_0 (active)
  xrender @0.9.0_0 (active)
  zlib @1.2.3_1 (active)
```

```
%% port -v installed atlas
```

```
The following ports are currently installed:
  atlas @3.8.3_4+gcc44-gcc43 (active) platform='darwin 10' archs='x86_64'
```

3.1.14. outdated

The outdated action checks your installed ports against the current ports tree to see if updated Portfiles have been released since your ports were installed. Note that you will not see new versions unless you have updated your ports tree using selfupdate or sync.

```
%% port outdated
```

```
apr                             1.2.8_0 < 1.2.9_0
autoconf                        2.61_0 < 2.61_1
gimp                            2.2.14_0 < 2.2.16_0
libtool                         1.5.22_0 < 1.5.24_0
pkgconfig                       0.21_0 < 0.22_0
```

3.1.15. upgrade

The upgrade action upgrades installed ports and their dependencies when a Portfile in the repository has been updated after a port was installed.

```
%% sudo port upgrade gnome
```

If you wish not to upgrade a port's dependencies, use the -n switch. Note that this will often cause problems.

```
%% sudo port -n upgrade gnome
```

If you'd like to upgrade all outdated ports, use this command.

```
%% sudo port upgrade outdated
```

> **Note**
>
> The upgrade action by default does not uninstall an upgraded port —it deactivates it. See section Port Images, and also Destroot and Activate phases in Port Phases. If you wish to uninstall the old version, use the -u option.
>
> ```
> %% sudo port -u upgrade vile
> ```

3.1.16. dependents

The dependents action reports what ports depend upon a given port, if any. MacPorts learns about dependents during port installation, so uninstalled ports will always report that there are no dependents.

```
%% port dependents openssl
```

```
neon depends on openssl
gnome-vfs depends on openssl
libdap depends on openssl
```

3.1.17. livecheck

The livecheck action checks to see if the application corresponding to a given port has been updated at the developer's download site. It's especially useful for port maintainers, but others may also wish to see if a port has the latest

available distribution source. See section Livecheck for more information.

```
%% port livecheck rrdtool
```

```
rrdtool seems to have been updated (port version: 1.2.23, new version: 1.3beta1)
```

Note

If livecheck finds no higher version at the port's download site, it prints nothing. The option `-d` (debug) may be used for detailed livecheck processing information.

3.1.18. lint

The lint action checks if the Portfile conforms to the MacPorts standards specified in Portfile Development.

If a Portfile validates fine the following message is shown.

```
%% port lint chemtool
```

```
--->  Verifying Portfile for chemtool
--->  0 errors and 0 warnings found.
```

Otherwise the warnings and errors are listed.

```
%% port lint KeyArcher
```

```
--->  Verifying Portfile for KeyArcher
Warning: Line 2 should be a newline (after RCS tag)
Warning: Line 5 has trailing whitespace before newline
Error: Missing required variable: platforms
--->  1 errors and 2 warnings found.
```

3.2. Port Variants

Variants are a way for port authors to provide options for a port that may be chosen during the port install. To display the available variants, if any, use this command:

```
%% port variants fetchmail
```

```
fetchmail has the variants:
        universal
        ssl: Support secure connections using OpenSSL
        fetchmailconf: Install a graphical configurator
        ntlm: Enable NTLM authentication
```

Note

In some ports, you may find variants for which descriptions have not yet been written.

3.2.1. Invoking Variants

A variant can only be invoked when a port is installed. After you have determined what variants a given port has, if any, you may install a port using the variant as shown.

```
%% sudo port install fetchmail +ssl
```

Port variant execution may be verified using the **port** command with the verbose switch.

```
%% sudo port -v install fetchmail +ssl
```

When a port is installed using a valid variant and specified correctly, the verbose output will contain:

```
DEBUG: Executing variant ssl provides ssl
```

3.2.2. Negating Default Variants

For an explanation of default variants see Port Variants. Default variants are optional, and not all ports using variants have them. For ports with default variants, you may install a port without them by negating default variants using "-" as shown.

```
%% sudo port install fetchmail -ssl
```

To verify that you have properly negated a default variant, you may want to use verbose mode. But negated variants are simply not reported in any way by the **port** command, as if they did not exist. You will know you have successfully negated the default variant in the example above if you do ***not*** see this line in the verbose output.

```
DEBUG: Executing variant ssl provides ssl
```

3.3. Common Tasks

This section lists common operations you may want to perform when managing a MacPorts installation. Some commands are described in further details elsewhere in the guide.

Mind the 'sudo' for some of the subsequent examples, which is necessary if you have a root-MacPorts-installation.

3.3.1. Updating your ports tree

is essential to stay up-to-date with MacPorts:

```
%% sudo port selfupdate
```

```
Password:
--->  Updating the ports tree
--->  Updating MacPorts base sources using rsync
MacPorts base version 1.9.1 installed,
MacPorts base version 1.9.1 downloaded.
--->  MacPorts base is already the latest version

The ports tree has been updated. To upgrade your installed ports, you should run
  port upgrade outdated
```

3.3.2. Show all ports which actually need updating

is often useful, in case you don't have time to wait for port upgrades (Every port needs to be downloaded, configured, built and installed, which can be - depending on your systems resources - a very time consuming procedure.):

```
%% port outdated
```

```
The following installed ports are outdated:
makedepend                      1.0.2_0 < 1.0.3_0
Xft2                            2.1.14_0 < 2.2.0_0
xorg-bigreqsproto               1.1.0_0 < 1.1.1_0
xorg-compositeproto             0.4.1_0 < 0.4.2_0
xorg-damageproto                1.2.0_0 < 1.2.1_0
xorg-fixesproto                 4.1.1_0 < 4.1.2_0
xorg-libXdmcp                   1.0.3_0 < 1.1.0_0
xorg-libXmu                     1.0.5_0 < 1.1.0_0
xorg-libXScrnSaver              1.2.0_0 < 1.2.1_0
xorg-randrproto                 1.3.1_0 < 1.3.2_0
xorg-scrnsaverproto             1.2.0_0 < 1.2.1_0
xorg-xcmiscproto                1.2.0_0 < 1.2.1_0
xpm                             3.5.8_0 < 3.5.9_0
xorg-util-macros                1.10.1_0 < 1.11.0_0
```

3.3.3. Upgrading outdated ports

is usually done with the following command:

```
%% sudo port upgrade outdated
```

In case you want to upgrade only a specific port (for whatever reason):

```
%% sudo port upgrade makedepend
```

```
Password:
--->  Computing dependencies for makedepend
--->  Fetching makedepend
--->  Attempting to fetch makedepend-1.0.3.tar.bz2 from http://lil.fr.distfiles.macports.org/makedepend
--->  Verifying checksum(s) for makedepend
--->  Extracting makedepend
--->  Configuring makedepend
--->  Building makedepend
--->  Staging makedepend into destroot
--->  Computing dependencies for makedepend
--->  Installing makedepend @1.0.3_0
--->  Deactivating makedepend @1.0.2_0
--->  Activating makedepend @1.0.3_0
--->  Cleaning makedepend
```

3.3.4. Removing inactive version(s) of upgraded port(s)

makes sense if you are sure that you won't change back to the older version(s) (which could be necessary in case of

newly introduced bugs in the upgraded port version):

```
%% port installed inactive
```

```
The following ports are currently installed:
  makedepend @1.0.2_0
```

```
%% sudo port uninstall inactive
```

```
Password:
--->  Uninstalling makedepend @1.0.2_0
```

Of course one could also select only a specific inactive port, but that requires to specify the exact version:

```
%% sudo port uninstall makedepend @1.0.2_0
```

```
Password:
--->  Uninstalling makedepend @1.0.2_0
```

3.3.5. Finding ports depending on a certain port

can sometimes be very helpful (example: find ports depending on 'xorg-util-macros'):

```
%% port echo depends:xorg-util-macros
```

```
mkfontscale
xorg-libAppleWM
xorg-libX11
xorg-libXaw
xorg-libXevie
xorg-libXp
xorg-libXt
xorg-server
xorg-server-devel
xscope
xtrap
```

3.3.6. Finding leaves (nondependent ports)

can be sometimes worthwhile, since often additional ports get installed when a specific port is installed. One can imagine that if one keeps installing and uninstalling ports for some time the MacPorts tree can be swamped with ports actually not necessary anymore. Leaves can be found by issuing the command:

```
%% port echo leaves
```

```
aalib                           @1.4rc5_4
autoconf                        @2.68_0
autoconf263                     @2.63_0
automake                        @1.11.1_0
cmake                           @2.8.2_4
coreutils                       @8.5_0
```

The following command delivers a similar result:

```
%% port installed leaves
```

```
  aalib @1.4rc5_4 (active)
  autoconf @2.68_0 (active)
  autoconf263 @2.63_0 (active)
  automake @1.11.1_0 (active)
  cmake @2.8.2_4 (active)
  coreutils @8.5_0 (active)
```

These leaves may be wanted, but could also be unwanted ports. (See the following entry!)

3.3.7. Keep your MacPorts installation lean by defining leaves as requested ports

Well, before we come to the procedure of defining your requested ports, let's have a look at a typical scenario where you want to understand what is actually installed and what is on the other hand truly necessary for your system. Say checking leaves of your MacPorts installation gives this output:

```
%% port echo leaves
```

```
aalib                           @1.4rc5_4
autoconf                        @2.68_0
autoconf263                     @2.63_0
```

```
automake                    @1.11.1_0
cmake                       @2.8.2_4
coreutils                   @8.5_0
physfs                      @2.0.1_0
```

Now it is up to the user to decide what's needed and what is not. Let's say one certainly would want ports starting with autoconf up to coreutils, since they are often used in port installation. Set those as requested ports using:

```
%% sudo port setrequested autoconf autoconf263 automake cmake coreutils
```

Say port aalib is not needed anymore, while physfs seems odd and since you don't know what to make of it, you need to check out what it actually is and which ports needed its installation at some time in the past.

```
%% port info physfs
```

```
physfs @2.0.1 (devel)
Variants:             debug, universal

Description:          PhysicsFS is a library to provide abstract access to various
                      archives. It is intended for use in video games, and the design
                      was somewhat inspired by Quake 3's file subsystem.
Homepage:             http://icculus.org/physfs/

Build Dependencies:   cmake
Library Dependencies: zlib
Platforms:            darwin
License:              unknown
Maintainers:          nomaintainer@macports.org
```

Well, hmm, you certainly didn't want to install Quake 3 in the past. So, let's find out which ports actually depend on it, perhaps that might ring a bell:

```
%% port echo depends:physfs
```

```
fbg
libsdl_sound
lincity-ng
netpanzer
```

No, you didn't install netpanzer either, but - see there - fbg (The classic Falling Block Game!) was once on your agenda. But unfortunately that port was broken (see issue #24641) when you were trying to install it... Before you ran into the issue, of course, fbg's dependencies were resolved by MacPorts, which is why port physfs is still present on your system. Uff, great, confusion resolved. :-)

When you've step-by-step figured out which ports you want to keep on your system and have set them as requested, you'll have a list of unnecessary ports, which might be eventually as short as this:

```
%% port echo leaves
```

```
aalib                       @1.4rc5_4
physfs                      @2.0.1_0
```

Get rid of them by issuing the command:

```
%% sudo port uninstall leaves
```

```
--->  Deactivating physfs @2.0.1_0
--->  Cleaning physfs
--->  Uninstalling physfs @2.0.1_0
--->  Cleaning physfs
--->  Deactivating aalib @1.4rc5_4
--->  Uninstalling aalib @1.4rc5_4
```

Find all installed ports you previously set as requested using:

```
%% port installed requested
```

Checking the list of leaves from time to time will help to keep your system free of too much "garbage". However, keep in mind that some ports may be dependencies only during the installation of a port (library dependencies, installation or configuration tools, etc.). So, be careful when removing seemingly unnecessary leaves, otherwise you'll see them being installed again and again.

3.4. Port Binaries

MacPorts can pre-compile ports into binaries so applications need not be compiled when installing on a target system. MacPorts may create two types of binaries: archives and packages.

3.4.1. Binary Archives

Binary archives can only be used on a target system running MacPorts. Binary archives allow MacPorts utilities to begin installation after the destroot phase and install and activate a port on the target system. Binary archives are created whenever a port is installed, and can also be downloaded from a server.

```
%% sudo port -d archive logrotate
```

Debug output is shown below.

```
--->  Creating logrotate-3.7_0.i386.tbz2

[ ... trimmed ... ]

DEBUG:
./
./+COMMENT
./+CONTENTS
./+DESC
./+PORTFILE
./+STATE
./opt/
./opt/local/
./opt/local/etc/
./opt/local/etc/logrotate.conf
./opt/local/man/
./opt/local/man/man8/
./opt/local/man/man8/logrotate.8
./opt/local/sbin/
./opt/local/sbin/logrotate
--->  Archive logrotate-3.7_0.i386.tbz2 packaged
DEBUG: Executing archive_finish
--->  Archive for logrotate 3.7_0 packaged
```

Binary archive files are placed in `${prefix}/var/macports/software/`. The archive file type is set in the `macports.conf` file. The default format is .tbz2; other options are: tar, tbz, tbz2, tgz, tlz, txz, xar, zip, cpgz, cpio.

3.4.2. Binary Packages

Binary packages are standalone binary installers that are precompiled; they do not require MacPorts on the target system. Binary files created with MacPorts may be either .pkg (Mac OS X Installer Packages), or RPM (RPM Package Manager) format. MacPorts may also process a .pkg package into a Mac OS X .dmg disk image file. You may create binary packages with the **port** command as shown in these examples.

```
%% sudo port pkg pstree
```

You may create a Mac OS X .dmg disk image file as shown.

```
%% sudo port dmg pstree
```

You can also create a metapackage containing packages of a port and all its dependencies:

```
%% sudo port mpkg gimp2
```

Just as with a single package, a metapackage can also be wrapped in a .dmg.

```
%% sudo port mdmg gimp2
```

You may compile a port into an RPM file as shown, in order to install it onto a target that has RPM utilities or a full package management system that can install RPMs.

```
%% sudo port rpm pstree
```

All packages are placed in a port's work directory.

3.4.3. Port Source Packages

Source packages are bundles consisting of a `Portfile`, patches if needed, and any other files required to install the port. Port source packages are mainly used by developers of package management and port submission frameworks. Port source packages may be in either .portpkg (XAR) or .nosrc.rpm (SRPM) format.

Chapter 4. Portfile Development

This chapter covers a brief introduction to Portfiles, how to create a local Portfile repository for development, and creating Portfiles.

4.1. Portfile Introduction

A MacPorts Portfile is a TCL script that usually contains only the simple keyword/value combinations and Tcl extensions as described in the Portfile Reference chapter, though it may also contain arbitrary TCL code. Every port has a corresponding Portfile, but Portfiles do not completely define a port's installation behavior since MacPorts base has default port installation characteristics coded within it. Therefore Portfiles need only specify required options, though some ports may require non-default options.

A common way for Portfiles to augment or override MacPorts base default installation phase characteristics is by using `Portfile` phase declaration(s). If you use Portfile phase declaration(s), you should know how to identify the "global" section of a Portfile. Any statements not contained within a phase declaration, no matter where they are located in a Portfile, are said to be in the global section of the Portfile; therefore the global section need not be contiguous. Likewise, to remove statements from the global section they must be placed within a phase declaration.

The main phases you need to be aware of when making a Portfile are these:

- Fetch
- Extract
- Patch
- Configure
- Build
- Destroot

The default installation phase behavior performed by the MacPorts base works fine for applications that use the standard **configure**, **make**, and **make install** steps, which conform to phases configure, build, and destroot respectively. For applications that do not conform to this standard behavior, any installation phase may be augmented using pre- and/or post- phases, or even overridden or eliminated. See Example Portfiles below.

> **Note**
>
> For a detailed description of all port phases, see the Portfile Reference chapter.

4.2. Creating a Portfile

Here we list the individual Portfile components for an application that conforms to the standard **configure**, **make**, and **make install** steps of most open source application installs.

1. Modeline

   This should be the first line of a Portfile. It sets the correct editing options for vim and emacs. See Port Style for more information. Its use is optional and up to the port maintainer.

   ```
   # -*- coding: utf-8; mode: tcl; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- vim:fenc=u
   ```

2. Subversion ID tag line

   This must be a Portfile's second line (or the first, if a modeline is not used). When a port is committed to the repository, ID tags are expanded to include the filename and the revision number, date and time, and author of the last commit.

   ```
   # $Id$
   ```

3. PortSystem line

   This statement is required for all ports.

   ```
   PortSystem          1.0
   ```

4. Port name

   ```
   name                rrdtool
   ```

5. Port version

   ```
   version             1.2.23
   ```

6. Port categories

   A port may belong to more than one category, but the first (primary) category should match the directory name in the ports tree where the Portfile is to reside.

```
categories          net
```

7. Platform statement

```
platforms           darwin
```

8. Port maintainers

   A port's maintainers are the people who have agreed to take responsibility for keeping the port up-to-date. The `maintainers` keyword lists the maintainers' email addresses, preferrably in the obfuscated form which hides them from spambots. For more, see the full explanation of the maintainers keyword in the Global Keywords section of the Portfile Reference chapter.

```
maintainers         jdoe \
                    example.org:julesverne
```

9. Port description

```
description         Round Robin Database
```

10. Port long_description

```
long_description    RRDtool is a system to store and display time-series \
                    data
```

11. A port's application homepage

```
homepage            http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/
```

12. A port's download URLs

```
master_sites        http://oss.oetiker.ch/rrdtool/pub/ \
                    ftp://ftp.pucpr.br/rrdtool/
```

13. Port checksums

    The checksums specified in a Portfile are checked with the fetched tarball for security. For the best security, use md5, sha1, and rmd160 checksum types.

```
checksums           md5     dafa161bc9c61e57636a6085c87c1fe8 \
                    sha1    5da610e1c8bc01b80abc21ab9e98e004363b429c \
                    rmd160  0c1147242adf476f5e93f4d59b553ee3ea378b23
```

    To find the correct checksums for a port's distribution file, follow this example:

```
%% md5 rrdtool-1.2.23.tar.gz
%% openssl sha1 rrdtool-1.2.23.tar.gz
%% openssl rmd160 rrdtool-1.2.23.tar.gz
```

```
MD5 ( ... rrdtool-1.2.23.tar.gz) = dafa161bc9c61e57636a6085c87c1fe8

SHA1( ... rrdtool-1.2.23.tar.gz)= 5da610e1c8bc01b80abc21ab9e98e004363b429c

RIPEMD160( ... rrdtool-1.2.23.tar.gz)= 0c1147242adf476f5e93f4d59b553ee3ea378b23
```

14. Port dependencies

    A port's dependencies are ports that must be installed before another port is installed.

```
depends_lib         port:perl5.8 \
                    port:tcl \
                    port:zlib
```

15. Port configure arguments (optional)

```
configure.args      --enable-perl-site-install \
                    --mandir=${prefix}/share/man
```

4.3. Example Portfiles

In this section we begin by taking a look at a complete simple Portfile; then we see how to augment default phases by defining pre- and post- phases, how to override default phases, and finally how to eliminate port phases.

4.3.1. A Basic Portfile

```
# $Id$
```

```
PortSystem          1.0

name                rrdtool
version             1.2.23
categories          net
platforms           darwin
license             GPL
maintainers         julesverne
description         Round Robin Database
long_description    RRDtool is a system to store and display time-series data
homepage            http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/
master_sites        http://oss.oetiker.ch/rrdtool/pub/ \
                    ftp://ftp.pucpr.br/rrdtool/

checksums           md5     dafa161bc9c61e57636a6085c87c1fe8 \
                    sha1    5da610e1c8bc01b80abc21ab9e98e004363b429c \
                    rmd160  0c1147242adf476f5e93f4d59b553ee3ea378b23

depends_lib         port:perl5.8 \
                    port:tcl \
                    port:zlib

configure.args      --enable-perl-site-install \
                    --mandir=${prefix}/share/man
```

4.3.2. Augment Phases Using pre- / post-

To augment a port's installation phase, and not override it, you may use pre- and post- installation phases as shown in this example.

```
post-destroot {
    # Install example files not installed by the Makefile
    file mkdir ${destroot}${prefix}/share/doc/${name}/examples
    file copy ${worksrcpath}/examples/ \
        ${destroot}${prefix}/share/doc/${name}/examples
}
```

4.3.3. Overriding Phases

To override the automatic MacPorts installation phase processing, define your own installation phases as shown in this example.

```
destroot {
    xinstall -m 755 -d ${destroot}${prefix}/share/doc/${name}
    xinstall -m 755 ${worksrcpath}/README ${destroot}${prefix}/share/doc/${name}
}
```

4.3.4. Eliminating Phases

To eliminate a default phase, simply define a phase with no contents as shown.

```
build {}
```

> **Note**
>
> Because many software packages do not use `configure`, a keyword is provided to eliminate the `configure` phase. Another exception is the `destroot` phase may not be eliminated. See the chapter Portfile Reference for full information.

4.3.5. Creating a StartupItem

Startupitems may be placed in the global section of a Portfile.

```
startupitem.create      yes
startupitem.name        nmicmpd
startupitem.executable  "${prefix}/bin/nmicmpd"
```

Startupitems keywords may also be used within a variant definition to make their installation conditional.

```
variant server {
    startupitem.create  yes
    startupitem.start   "${prefix}/share/${name}/vm-pop3d.init start"
    startupitem.stop    "${prefix}/share/${name}/vm-pop3d.init stop"

}
```

4.4. Port Variants

Variants are a way for port authors to provide options that may be invoked at install time. They are declared in the global section of a Portfile using the "variant" keyword, and should include carefully chosen variant descriptions.

4.4.1. Example Variants

The most common actions for user-selected variants is to add or remove dependencies, configure arguments, and build arguments according to various options a port author wishes to provide. Here is an example of several variants that modify depends_lib and configure arguments for a port.

```
variant fastcgi description {Add fastcgi binary} {
    configure.args-append \
            --enable-fastcgi \
            --enable-force-cgi-redirect \
            --enable-memory-limit
}

variant gmp description {Add GNU MP functions} {
    depends_lib-append port:gmp
    configure.args-append --with-gmp=${prefix}

}

variant sqlite description {Build sqlite support} {
    depends_lib-append \
        port:sqlite3
    configure.args-delete \
        --without-sqlite \
        --without-pdo-sqlite
    configure.args-append \
        --with-sqlite \
        --with-pdo-sqlite=${prefix} \
        --enable-sqlite-utf8
}
```

Note

Variant names may contain only the characters A-Z, a-z, and the underscore character "_". Therefore, take care to never use hyphens in variant names.

In the example variant declaration below, the configure argument `--without-x` is removed and a number of others are appended.

```
variant x11 description {Builds port as an X11 program with Lucid widgets} {
    configure.args-delete   --without-x
    configure.args-append   --with-x-toolkit=lucid \
                            --without-carbon \
                            --with-xpm \
                            --with-jpeg \
                            --with-tiff \
                            --with-gif \
                            --with-png
    depends_lib-append      lib:libX11:XFree86 \
                            lib:libXpm:XFree86 \
                            port:jpeg \
                            port:tiff \
                            port:libungif \
                            port:libpng
}
```

4.4.2. Variant Actions in a Phase

If a variant requires options in addition to those provided by keywords using -append and/or -delete, in other words, any actions that would normally take place within a port installation phase, do not try to do this within the variant declaration. Rather, modify the behavior of any affected phases when the variant is invoked using the variant_isset keyword.

```
post-destroot {
    xinstall -m 755 -d ${destroot}${prefix}/etc/
    xinstall ${worksrcpath}/examples/foo.conf \
        ${destroot}${prefix}/etc/


    if {[variant_isset carbon]} {
        delete ${destroot}${prefix}/bin/emacs
```

```
        delete ${destroot}${prefix}/bin/emacs-${version}
    }
}
```

4.4.3. Default Variants

Variants are used to specify actions that lie outside the core functions of an application or port, but there may be some cases where you wish to specify these non-core functions by default. For this purpose you may use the keyword default_variants.

```
default_variants    +foo +bar
```

Note

The default_variant keyword may only be used in the global Portfile section.

4.5. Patch Files

Patch files are files created with the Unix command **diff** that are applied using the command **patch** to modify text files to fix bugs or extend functionality.

4.5.1. Creating Portfile Patches

If you wish to contribute modifications or fixes to a Portfile, you should do so in the form of a patch. Follow the steps below to create Portfile patch files

1. Make a copy of the Portfile you wish to modify; both files must be in the same directory, though it may be any directory.

   ```
   %% cp -p Portfile Portfile.orig
   ```

2. Edit the file to make it as you want it to be after it is fetched.

3. Now use the Unix command **diff -u** to create a "unified" diff patch file. Put the name of the port in the patchfile, for example, Portfile-rrdtool.diff.

   ```
   %% diff -u Portfile.orig Portfile > Portfile-rrdtool.diff
   ```

4. A patch file that is a "unified" diff file is the easiest to interpret by humans and this type should always be used for ports. The Portfile patch below will change the version and checksums when applied.

   ```
   --- Portfile.orig        2007-07-25 18:52:12.000000000 -0700
   +++ Portfile    2007-07-25 18:53:35.000000000 -0700
   @@ -2,7 +2,7 @@
    PortSystem          1.0
    name                foo

   -version             1.3.0
   +version             1.4.0
    categories          net
    maintainers         nomaintainer
    description         A network monitoring daemon.
   @@ -13,9 +13,9 @@

    homepage            http://rsug.itd.umich.edu/software/${name}

    master_sites        ${homepage}/files/
   -checksums           md5 01532e67a596bfff6a54aa36face26ae
   +checksums           md5 f0953b21cdb5eb327e40d4b215110b71
    extract.suffix      .tgz
    platforms           darwin
   ```

Now you may attach the patch file to a MacPorts Trac ticket for the port author to evaluate.

4.5.2. Creating Source Code Patches

Necessary or useful patches to application source code should generally be sent to the application developer rather than the port author so the modifications may be included in the next version of the application.

Generally speaking, you should create one patch file for each file to be patched. Patchfile filenames should uniquely distinguish the file and generally be of the form `patch-<directory>-<filename>.diff`, as shown in this example: `patch-src-Makefile.in.diff`.

You may use patch files that patch multiple files under these conditions:

- You find existing patch files that do so.
- If fixing a particular problem or bug requires changes in multiple files -in those cases the patch filename should reference the problem or bug, for example: `patch-<destroot_variable_fix>.diff`

To create a patch to modify a single file, follow the steps below.

1. Locate the file you wish to patch in its original location within the unpacked source directory and make a duplicate of it.

```
%% cd ~/Downloads/foo-1.34/src
%% cp -p Makefile.in Makefile.in.orig
```

2. Edit the file and modify the text to reflect your corrections.

3. Now **cd** to the top-level directory of the unpacked source, and use the Unix command **diff -u** to create a "unified" diff patch file.

```
%% cd ~/Downloads/foo-1.34
%% diff -u src/Makefile.in.orig src/Makefile.in > patch-src-Makefile.in.diff
```

You should execute the **diff** command from the top-level directory of the unpacked source code, because during the patch phase MacPorts by default uses the patch argument `-p0`, which does not strip prefixes with any leading slashes from file names found in the patch file (as opposed to `-p1` that strips one, etc), and any path not relative to the top-level directory of the unpacked source will fail during the patch phase.

> **Note**
>
> If you find an existing source file patch you wish to use that contains leading path information (diff was executed from a directory higher than the top-level source directory), you will need to use the patch phase keyword `patch.pre_args` to specify a `-px` value for how many prefixes with leading slashes are to be stripped off.

4. A patch file that is a "unified" diff file is the easiest to interpret by humans and this type should always be used for ports. See the example below where a patch adds `DESTDIR` support to a `Makefile.in` file.

```
--- src/Makefile.in.orig   2007-06-01 16:30:47.000000000 -0700
+++ src/Makefile.in       2007-06-20 10:10:59.000000000 -0700
@@ -131,23 +131,23 @@
        $(INSTALL_DATA)/gdata $(INSTALL_DATA)/perl

 install-lib:
-       -mkdir -p $(INSTALL_LIB)
+       -mkdir -p $(DESTDIR)$(INSTALL_LIB)
        $(PERL) tools/install_lib -s src -l $(INSTALL_LIB) $(LIBS)
-       cp $(TEXT) $(INSTALL_LIB)/
+       cp $(TEXT) $(DESTDIR)$(INSTALL_LIB)/
```

5. Place the patch `patch-src-Makefile.in.diff` in the directory `${portpath}/files` and use it in a port using the `patchfiles` keyword. `${portpath}` may be in a local Portfile repository during development, or `files/` may be in a port's `${portpath}` in the global MacPorts repository.

```
patchfiles          patch-src-Makefile.in.diff
```

4.5.3. Manually Applying Patches

MacPorts applies patch files automatically, but you may want to know how to apply patch files manually if you want to test patch files you have created or you wish to apply uncommitted Portfile patches.

1. Change to the directory containing the file to be patched. In this example, we'll apply a Portfile patch to the postfix port.

```
%% cd $(port dir postfix)
```

2. Now apply the patch from your Downloads folder, or wherever you put it. The patchfile knows the name of the file to be patched.

```
%% patch -p0 < ~/Downloads/Portfile-postfix.diff
```

```
patching file Portfile
```

4.6. Local Portfile Repositories

To create and test Portfiles that are not yet committed to Subversion, you may create a local Portfile repository as shown. Replace the hypothetical user `julesverne` with your username in the example below.

1. Open the `sources.conf` file in a text editor. For example, to open it into TextEdit:

```
%% open -e ${prefix}/etc/macports/sources.conf
```

2. Insert a URL pointing to your local repository location before the rsync URL as shown.

```
file:///Users/julesverne/ports
rsync://rsync.macports.org/release/ports [default]
```

> **Note**
>
> The file URL should always appear before the rsync URL so that local Portfiles can be tested that are duplicated in the MacPorts tree, because the **port** command will always operate on the first Portfile it encounters.

3. Place the Portfiles you create inside a directory whose name matches the port, which should in turn be placed inside a directory that reflects the port's primary category (the first category entry in the Portfile). For example, to create the directory for a hypothetical port "bestevergame" and to begin editing its Portfile in TextEdit, you can use these commands:

```
%% mkdir -p ~/ports/games/bestevergame
%% cd ~/ports/games/bestevergame
%% touch Portfile
%% open -e Portfile
```

   See other sections in the Guide for help writing Portfiles. If you've already written the Portfile elsewhere, you can instead copy the Portfile into this directory.

4. If your Portfile needs to apply any patches to the port's source files, create a `files` directory and place the patchfiles in it, and reference the patchfiles in your Portfile, as explained in Creating Source Code Patches.

5. After you create or update your Portfile, use the MacPorts **portindex** command in the local repository's directory to create or update the index of the ports in your local repository.

```
%% cd ~/ports
%% portindex
```

```
Creating software index in /Users/julesverne/ports
Adding port games/bestevergame

Total number of ports parsed:   1
Ports successfully parsed:      1
Ports failed:                   0
```

Once the local port is added to the `PortIndex`, it becomes available for searching or installation as with any other Portfile in the MacPorts tree:

```
%% port search bestever
```

```
bestevergame @1.1 (games)
    The Best Ever Game
```

4.7. Portfile Best Practices

This section contains practical guidelines for creating Portfiles that install smoothly and provide consistency between ports. The following sections are on the TODO list.

4.7.1. Port Style

Portfiles may be thought of as a table of keys and values in two columns separated by spaces (not tabs), so you should set your editor to use soft tabs, which are tabs emulated by spaces. By default, the top line of all Portfiles should use a modeline that defines soft tabs for the vim and emacs editors as shown.

```
# -*- coding: utf-8; mode: tcl; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- vim:fenc=utf-8:
```

The left column should consist of single words, and will be separated from the more complex right side by spaces in multiples of four. Variable assignments and variant declarations are exceptions, and may be considered a single word on the left side, with a single space between words.

```
set libver "8.5"



variant mysql5 { ... }
```

Frequently multiple items are necessary in the second column. For example, to set multiple source download locations

Frequently multiple items are necessary in the second column. For example, to set multiple source download locations, multiple "master_sites" must be defined. Unless the second column items are few and short you should place each additional item on a new line and separate lines with a backslash. Indent the lines after the first line to make it clear the items are second column values and also to emphasize the unity of the block.

```
destroot.keepdirs    ${destroot}${prefix}/var/run \
                     ${destroot}${prefix}/var/log \
                     ${destroot}${prefix}/var/cache/mrtg
```

4.7.2. Don't Overwrite Config Files

TODO:

4.7.3. Install Docs and Examples

TODO:

4.7.4. Provide User Messages

TODO:

4.7.5. Use Variables

TODO: Set variables so changing paths may be done in one place; use them anytime it makes updates simpler: distname ${name}-src-${version}

4.7.6. Renaming or replacing a port

If there is the need to replace a port with another port or a renaming is necessary for some reason, the port should be marked as “replaced_by”.

As an illustration of a typical workflow the port 'skrooge-devel' shall be taken. This port had been used for testing new versions of skrooge, but it turned out to have become unnecessary due to the fact that skrooge's developers currently prefer a distribution via port 'skrooge' instead.

Skrooge's original devel port file looked like this:

```
# -*- coding: utf-8; mode: tcl; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4; truncate-lines: t
# $Id$

PortSystem          1.0
PortGroup           kde4    1.1

fetch.type          svn
svn.url             svn://anonsvn.kde.org/home/kde/trunk/extragear/office/skrooge
svn.revision        1215845

name                skrooge-devel
version             0.8.0-${svn.revision}

categories          kde finance
maintainers         mk pixilla openmaintainer
description         Skrooge
long_description    Personal finance management tool for KDE4, with the aim of being highly intuitive, whi
                    providing powerful functions such as reporting (including graphics), persistent \
                    Undo/Redo, encryption, and much more...

conflicts           skrooge

platforms           darwin
license             GPL-3

homepage            http://skrooge.org
master_sites        http://skrooge.org/files/

livecheck.type      none

distname            skrooge

depends_lib-append  port:kdelibs4 \
                    port:libofx \
                    port:qca-ossl \
                    port:kdebase4-runtime \
                    port:oxygen-icons
```

The following steps have to be taken to ensure a smooth transition for a MacPorts user updating his local installation using “sudo port upgrade”:

1. add the line “replaced_by foo” where foo is the port this one is replaced by; when a user upgrades this port, MacPorts will instead install the replacement port

```
replaced_by         skrooge
```

2. increase the version, revision, or epoch, so that users who have this port installed will get notice in “port outdated” that they should upgrade it and trigger the above process

```
revision            1
```

3. clear distfiles (have a line reading only “distfiles”) so that no distfile is downloaded for this stub port

```
distfiles
```

4. delete master_sites since there aren't any distfiles to download

5. disable livecheck

```
livecheck.type      none
```

6. add a pre-configure block with a “ui_error” and “return -code error” explaining to users who try to install this port that the port has been replaced

```
pre-configure {
    ui_error "Please do not install this port since it has been replaced by 'skrooge'."
    return -code error
}
```

With above modifications the port file eventually looks like this:

```
# -*- coding: utf-8; mode: tcl; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4; truncate-lines: t
# $Id$

PortSystem          1.0

name                skrooge-devel
svn.revision        1215845
version             0.8.0-${svn.revision}
revision            1

replaced_by         skrooge

categories          kde finance
maintainers         mk pixilla openmaintainer
description         Skrooge
long_description    Personal finance management tool for KDE4, with the aim of being highly intuitive, whi
                    providing powerful functions such as reporting (including graphics), persistent \
                    Undo/Redo, encryption, and much more...

platforms           darwin
license             GPL-3

homepage            http://skrooge.org

livecheck.type      none

pre-configure {
    ui_error "Please do not install this port since it has been replaced by 'skrooge'."
    return -code error
}

distfiles
```

A user upgrading ports will experience for port 'skrooge-devel' the following:

```
%% sudo port upgrade skrooge-devel
```

```
--->  skrooge-devel is replaced by skrooge
--->  Computing dependencies for skrooge
--->  Fetching skrooge
--->  Verifying checksum(s) for skrooge
--->  Extracting skrooge
--->  Configuring skrooge
--->  Building skrooge
--->  Staging skrooge into destroot
```

```
--->  Staging skrooge into destroot
--->  Deactivating skrooge-devel @0.8.0-1215845_0
--->  Cleaning skrooge-devel
--->  Computing dependencies for skrooge
--->  Installing skrooge @0.8.0.6_0
--->  Activating skrooge @0.8.0.6_0
########################################################
# Don't forget that dbus needs to be started as the local
# user (not with sudo) before any KDE programs will launch
# To start it run the following command:
# launchctl load /Library/LaunchAgents/org.freedesktop.dbus-session.plist
########################################################

##################################################
#  Programs will not start until you run the command
#  'sudo chown -R $USER ~/Library/Preferences/KDE'
#  replacing $USER with your username.
##################################################
--->  Cleaning skrooge
```

In case a user actually tries to install the obsolete port 'skrooge-devel' it would be pointed out by an error message that this is impossible now:

```
%% sudo port install skrooge-devel
```

```
--->  Fetching skrooge-devel
--->  Verifying checksum(s) for skrooge-devel
--->  Extracting skrooge-devel
--->  Configuring skrooge-devel
Error: Please do not install this port since it has been replaced by 'skrooge'.
Error: Target org.macports.configure returned:
Log for skrooge-devel is at: /opt/local/var/macports/logs/_opt_local_var_macports_sources_rsync.macports.or
Error: Status 1 encountered during processing.
To report a bug, see <http://guide.macports.org/#project.tickets>
```

4.7.7. Removing a port

If a port has to be removed from MacPorts one should consider the hints concerning replacing it by some alternative port given above. If there is no replacement for it, insert a pre-configure block as described there to alert the user about why the port is not allowed for installation anymore.

It is recommended to wait about a year before the port directory is actually being removed from MacPorts' Subversion repository.

Chapter 5. Portfile Reference

This chapter serves as a reference for the major elements of a Portfile: port phases, dependencies, StartupItems, variables, keywords, and Tcl extensions.

5.1. Global Keywords

MacPorts keywords are used to specify required or optional items within a Portfile, or to override default options used by MacPorts base for individual ports. Keywords are to be used within the "global" and "variant" sections of Portfiles, and not within optional port phase declarations.

The global keywords listed below specify information for ports as a whole, whereas the keywords listed under a port phase specify information to be used during a particular installation phase.

PortSystem

The top line of every Portfile; it must be followed by a blank line. It defines which version of the Portfile interpreter will be used.

```
PortSystem          1.0
```

name

The name of the port. To avoid special interpretation by shells and the like, names should contain only alphanumeric characters, underscores, dashes or dots.

```
name                foo
```

version

The version of the ported software.

```
version             1.23.45
```

```
version             1.23.45
```

revision

Optional keyword (default is 0) that is used to track port revisions. It should not be incremented for port revisions unless it would benefit users to upgrade an installed port, and cleared when the port is updated to a newer version.

It should be used if a bug in the Portfile was found and all installations of this port have to be updated. If the change only affects new installations, there is no need to increase it.

```
revision            1
```

epoch

An optional keyword (default value is 0) that must be used when a port is updated to a version that is numerically less than the previous version, for example 1.10 -> 1.2 or 20070928 -> 1.0. Some Portfile authors have used large epoch values that look like a date, but there is no reason to do so. The epoch is simply an unsigned integer, and the only requirement is that it never be decreased.

```
epoch               1
```

Note

An epoch is not needed for most ports. If a port's version numbers advance in normal dotted-decimal sequence, there is no reason to add an epoch.

categories

The category under which the ported software falls. The first category should be the same as the directory within which the Portfile is stored; secondary and tertiary categories may be selected.

```
categories          net security
```

maintainers

A port's maintainers are the people who have agreed to take responsibility for keeping the port up-to-date. Most ports have only a single maintainer, but some ports have two or more comaintainers. The `maintainers` keyword lists the maintainers' email addresses, preferrably in the obfuscated form which hides them from spambots:

- For addresses in domain @macports.org, simply omit the domain name.
- For addresses in other domains, e.g. <account@example.org>, use the convention example.org:account to specify the address.

In the example below, the maintainer email addresses <jdoe@macports.org> and <julesverne@example.org> are hidden using these conventions.

```
maintainers         jdoe \
                    example.org:julesverne
```

Note

The address <nomaintainer> designates a port that is not maintained by anybody and may be modified by any committer. Feel free to claim maintainership of a nomaintainer port if desired. The address <openmaintainer> designates a port that has a maintainer who allows minor changes to be committed without his or her prior approval.

description

A one-sentence description of the ported software.

```
description         A classic shooter arcade game.
```

long_description

A long description of the ported software. Break long lines with escaped newlines.

```
long_description    A classic shooter arcade game derived from \
                    the game alien-munchers.  Not suitable for \
                    children under two years old.
```

homepage

Port application's homepage.

```
homepage            http://www.example.org/apps
```

platforms

The platforms on which the port has been tested.

```
platforms           darwin freebsd
```

supported_archs

The CPU architectures for which this port can be built. Archs currently supported by Mac OS X are: i386, ppc, ppc64, x86_64. If this option is not set, it is assumed that the port can build for all archs. If a port does not install any architecture-specific files, use the special value `noarch`.

If the building architecture isn't among supported_archs, port fails with an error message except when building on x86_64 (resp. ppc64) and supported_archs contains i386 (resp. ppc). In this case, the port will be built in 32 bit mode.

```
supported_archs           i386 ppc
```

```
supported_archs           noarch
```

license

The proper format for license consists of the license name, followed by a hyphen and number if indicating a specific version. A space should be placed between licenses if there is more than one that applies. If an element in the license list is itself a list, it is interpreted as offering a choice of any one of the licenses in the sub-list.

If the version number is a ".0" version, the ".0" should be omitted to make the version an integer. If the author gives the choice of using a given license or "any later version" of it, append a plus sign (+) to the version number. If the version specified in this case is also the earliest version, just leave out the version number entirely since it implies all versions.

```
license           GPL-3
```

```
license           {freetype GPL}
```

5.2. Global Variables

Global variables are variables available to any Portfile. For a list of additional variables available to ports that are assigned to a MacPorts Portgroup, see portgroup(7).

All of these variables except `prefix` are read-only!

prefix

Installation prefix, set at compile time and displayed in `${prefix}/etc/macports/macports.conf` —- may be overridden on a per-port basis, for example to install into a wholly-contained subdirectory of ${prefix}, but most ports should have no reason to do so.

Default: `/opt/local`

libpath

Path to the MacPorts TCL libraries.

portpath

Full path to the Portfile of the port being executed. Portfile repositories are defined in the file sources.conf.

Default: `${prefix}/var/macports/sources/rsync.macports.org/release/ports/<category>/<portname>/`

filesdir

Path to files directory relative to `${portpath}`.

Value: `files`

filespath

Full path to files directory.

Value: `${portpath}/${filesdir}`

workpath

Full path to work directory.

Value: `${portbuildpath}/work`

worksrcpath

Full path to extracted source code.

Value: `${workpath}/${worksrcdir}`

destroot

Full path into which software will be destrooted.

Value: `${workpath}/destroot`

distpath

Location to store downloaded distfiles.

Value: `${sysportpath}/distfiles/${dist_subdir}/`

install.user

The Unix user at the time of port installation.

install.group

The Unix group at the time of port installation.

os.platform

The underlying operating system platform (i.e. "darwin" on Mac OS X, "freebsd", etc.).

os.arch

The hardware architecture -- either "powerpc" or "i386".

os.version

The version number of the host operating system (i.e. "8.11.0" for Darwin 8.11.0 a.k.a. Mac OS X 10.4.11).

os.endian

Endianness of the processor -- either "big" (on PowerPC systems) or "little" (on Intel systems).

os.major

The major version number of the host operating system (i.e. "8" for Darwin 8.x).

5.3. Port Phases

5.3.1. Introduction

The MacPorts port installation process has a number of distinct phases that are described in detail in this section. The default scripts coded into MacPorts base performs the standard **configure**, **make**, and **make install** steps. For applications that do not conform to this standard, installation phases may be declared in a Portfile to augment or override the default behavior as described in the Portfile Development chapter.

fetch

Fetch the `${distfiles}` from `${master_sites}` and place it in `${prefix}/var/macports/distfiles/${name}`.

checksum

Compare `${checksums}` specified in a `Portfile` to the checksums of the fetched ${distfiles}.

extract

Unzip and untar the `${distfiles}` into the path ${prefix}/var/macports/build/..../work

patch

Apply optional patch files specified in `${patchfiles}` to modify a port's source code file(s).

configure

Execute the command **configure** in `${worksrcpath}`.

build

Execute `${build.cmd}` in `${worksrcpath}`.

test

Execute commands to run test suites bundled with a port.

destroot

Execute the command **make install** `DESTDIR=${destroot}`in `${worksrcpath}`.

> **Note**
>
> Using a `DESTDIR` variable is a part of standard GNU coding practices, and this variable must be supported in an application's install routines for MacPorts' destroot phase to work without manual Portfile scripting or source patching. Urge developers to fully support `DESTDIR` in their applications.

Understanding the destroot phase is critical to understanding MacPorts, because, unlike some port systems, MacPorts "stages" an installation into an intermediate location —not the final file destination. MacPorts uses the destroot phase to provide:

- Port uninstalls - a port's files may be cleanly uninstalled because all files and directories are recorded during install.
- Multiple port versions may be installed on the same host, since a port's files are not directly inserted into `${prefix}` but rather hard-linked into `${prefix}` from an intermediate location during a later activation phase.

Any empty directories in `${destroot}` upon completion of the destroot phase are removed unless a directory name is placed in the value field of the optional `destroot.keepdirs` keyword.

install

Archive a port's destrooted files into `${prefix}/var/macports/software`. See Port Images in the MacPorts Internals chapter for details.

activate

Extract the port's files from the archive in `${prefix}/var/macports/software` to their final installed locations, usually inside `${prefix}`.

5.3.2. Installation Phase Keywords

MacPorts keywords are used to specify required or optional items within a Portfile, or to override default options used by MacPorts base for individual ports. Keywords are to be used within the "global" and "variant" sections of Portfiles, and not within optional port phase declarations.

In other words, port phase keywords are not located within port phase declarations, but rather they ***refer*** to port phases and set options for those phases, and they take effect whether or not phase declarations have been explicitly defined in a Portfile.

5.3.2.1. Keyword List Modifiers (-append, -delete, -replace)

Keyword list modifiers are keywords that end in -append, -delete or -replace. Keywords that support list modifiers are identified under appropriate reference sections below.

-append adds a value to the keyword, -delete removes a previously added item. -replace treats the keyword value as a string and filters it through strsed using a given pattern.

Keyword list modifiers are most frequently used for these three purposes:

1. Preserve configure defaults set by a previously executed Portfile keyword or by MacPorts base

   MacPorts base sets the gcc compiler flags CFLAGS and LDFLAGS for all ports using `configure.cflags` and `configure.ldflags`, therefore to keep from overwriting the default compiler flags use `configure.cflags-append` and `configure.ldflags-append`.

   - `configure.cflags-append`
   - `configure.ldflags-append`

2. Preserve PortGroup Dependencies

   Ports in a PortGroup have default library dependencies set by MacPorts base. Therefore, never use `depends_lib` in ports belonging to a PortGroup or it will overwrite the default library dependencies. Instead, use `depends_lib-append`.

3. Add or Delete Items for Variants

When a variant requires more or fewer dependencies, distfiles, or patchfiles, when the variant is invoked you want to add or remove items to the appropriate keyword values list set in the global section of the Portfile. Use the appropriate keywords, for example:

- `depends_lib-append` or `depends_lib-delete` `depends_lib-replace`
- `distfiles-append` or `distfiles-delete` `distfiles-replace`
- `patchfiles-append` or `patchfiles-delete` `patchfiles-replace`

5.3.2.2. Keyword Argument Modifiers (.pre_args / .post_args)

Keywords that support pre_args and post_args are used to assemble command strings together in a row, as described in the reference sections below. But it should be noted that all keyword argument modifiers implicitly support keyword list modifiers. For example, the keyword `configure.pre_args` also supports `configure.pre_args-append` and `configure.pre_args-delete`.

5.3.3. Fetch Phase Keywords

The list of keywords related to the fetch phase.

master_sites

A list of URLs from which a port's `${distfiles}` may be retrieved.

Keyword values for `master_sites` may include predefined site lists known as "mirrors", such as sourceforge, gnu, etc. If the file(s) declared in `${distfiles}` are not successfully fetched after trying the `master_sites` values, the MacPorts Project svn server is always tried last before giving up.

For a complete list of mirrors and their list of sites, see the file `mirror_sites.tcl` located in `_resources/port1.0/fetch/` in the ports tree.

Note

If a `master_sites` keyword has multiple values, after any mirrors are expanded the list of sites is sorted by ping response times. The sites are then tried in sorted order until matching `${distfiles}` are found.

- Default: `none` (but the `macports_distfiles` mirror is always implicitly appended)
- Examples:

```
master_sites        http://www.example.org/files/ \
                    http://www.examplemirror.org/example_org/files/
```

You may also use mirror site lists predefined by MacPorts. Here the sourceforge, gnu, and freebsd mirrors are used.

```
master_sites        sourceforge gnu freebsd
```

When using mirror master_sites, the subdirectory `${name}` is checked on every mirror. If the mirror subdirectory does not match ${name}, then you may specify it using after the mirror separated by a colon.

```
master_sites        sourceforge:widget \
                    gnu:widget
```

For ports that must fetch multiple download files from different locations, you must label the files with tags and match the tags to a `distfiles` keyword. The format is `mirror:subdirectory:tag`.

In the example below, file_one.tar.gz is fetched from sourceforge mirrors in subdirectory `${name}`; file tagtwo.tar.gz is fetched from the gnu mirrors in subdirectory sources.

```
master_sites        sourceforge::tagone \
                    gnu:sources:tagtwo

distfiles           file_one.tar.gz:tagone \
                    file_two.tar.gz:tagtwo
```

master_sites.mirror_subdir

Subdirectory to append to all mirror sites for any list specified in `${master_sites}`.

- Default: `${name}`
- Example:

```
master_sites.mirror_subdir  magic
```

patch_sites

A list of sites from which a port's patchfiles may be downloaded, where applicable.

- Default: `${master_sites}`
- Example:

```
patch_sites         ftp://ftp.patchcityrepo.com/pub/magic/patches
```

patch_sites.mirror_subdir

Subdirectory to append to all mirror sites for any list specified in `${patch_sites}`.

- Default: `${name}`
- Example:

```
patch_sites.mirror_subdir   magic
```

distname

The name of the distribution filename, not including the extract suffix (see below).

- Default: `${name}-${version}`
- Example:

```
distname            ${name}
```

distfiles

The full distribution filename, including the extract suffix. Used to specify non-default distribution filenames; this keyword must be specified (and tags used) when a port has multiple download files (see master_sites).

- Default: `${distname}${extract.suffix}`
- Examples:

```
distfiles           ${name}-dev_src.tgz
```

```
distfiles           file_one.tar.gz:tagone \
                    file_two.tar.gz:tagtwo
```

dist_subdir

Create a sub-directory in distpath to store all fetched files.

- Default: `${name}`
- Example:

```
dist_subdir         vim${version}
```

worksrcdir

Sets the path to source directory relative to workpath. It can be used if the extracted source directory has a different name then the distfile. Also used if the source to be built is in a subdirectory.

- Default: `${distname}`
- Examples:

```
worksrcdir          ${name}-src-${version}
```

```
worksrcdir          ${distname}/src
```

5.3.3.1. Advanced Fetch Options

Some mirrors require special options for a resource to be properly fetched.

fetch.type

Change the fetch type. This is only necessary if a CVS, svn, git or hg checkout is be used. `standard` is used for a normal http or ftp fetch using `${distfiles}` and is used as default.

- Default: `standard`
- Values: `standard cvs svn git`
- Example:

```
fetch.type          svn
svn.url             svn://example.org
svn.revision        2100
```

fetch.user

HTTP or FTP user to fetch the resource.

- Default: none
- Example:

```
TODO: add example
```

fetch.password

HTTP or FTP password to fetch the resource.

- Default: none
- Example:

```
TODO: add example
```

fetch.use_epsv

Whether to use EPSV command for FTP transfers.

- Default: `yes`
- Example:

```
fetch.use_epsv      no
```

fetch.ignore_sslcert

Whether to ignore the host SSL certificate (for HTTPS).

- Default: `no`
- Example:

```
fetch.ignore_sslcert    yes
```

5.3.3.2. Fetch from CVS

CVS may be used as an alternative method of fetching distribution files using the keywords in this section. However, fetching via CVS may cause non-reproducible builds, so it is strongly discouraged.

The `cvs` fetch.type is used to fetch source code from a CVS repository.

cvs.root

Specify the url from which to fetch files.

- Default: none
- Example:

```
cvs.root            :pserver:anonymous@cvs.sv.gnu.org:/sources/emacs
```

cvs.password

Password to login to the CVS server.

- Default: none
- Example:

```
cvs.password        nice-password
```

cvs.tag

Optional for fetching with CVS, this specifies the code revision to checkout.

- Default: none
- Example:

```
cvs.tag             HEAD
```

cvs.date

A date that identifies the CVS code set to checkout.

- Default: none
- Example:

```
cvs.date            "12-April-2007"
```

cvs.module

A CVS module from which to check out the code.

- Default: none
- Example:

```
cvs.module          Sources
```

5.3.3.3. Fetch from Subversion

Subversion may be used as an alternative method of fetching distribution files using the keywords in this section. However, fetching via Subversion may cause non-reproducible builds, so it is strongly discouraged.

The `svn` fetch.type is used to fetch source code from an svn repository.

svn.url

This specifies the url from which to fetch files.

- Default: none
- Examples:

```
svn.url             http://www.domain.com/svn-repo/mydirectory
```

```
svn.url             svn://www.domain.com/svn-repo/mydirectory
```

svn.revision

Optional tag for fetching with Subversion, this specifies the peg revision to checkout; it corresponds to the @REV syntax of the svn cli.

- Default: none
- Example:

```
svn.revision        37192
```

svn.method

Optional tag for fetching with Subversion, this specifies whether to check out the code into a working copy, or just export it without the working copy metadata. An export is preferable because it takes half the disk space, but some software expects to be built in a working copy (for example because it wants to record the revision number into itself somewhere).

- Default: export
- Example:

```
svn.method          checkout
```

5.3.3.4. Fetch from Git

Git may be used as an alternative method of fetching distribution files using the keywords in this section. However, fetching via Git may cause non-reproducible builds, so it is strongly discouraged.

The `git` fetch.type is used to fetch source code from a git repository.

git.url

This specifies the url from which to fetch files.

- Default: none
- Examples:

```
git.url             git://git.kernel.org/pub/scm/git/git.git
```

```
git.url             http://www.kernel.org/pub/scm/git/git.git
```

git.branch

Optional tag for fetching with git, this specifies a branch (or other commit-ish) that git should checkout. Note that any branch besides HEAD should be prefixed by origin/.

- Default: none
- Example:

```
git.branch              72bf1c8
```

```
git.branch              origin/next
```

5.3.3.5. Fetch from Mercurial

Mercurial may be used as an alternative method of fetching distribution files using the keywords in this section. However, fetching via Mercurial may cause non-reproducible builds, so it is strongly discouraged.

The `hg` fetch.type is used to fetch fource code from a Mercurial repository.

hg.url

This specifies the url from which to fetch files.

- Default: none
- Examples:

```
hg.url              http://www.kernel.org/hg/index.cgi/linux-2.6/
```

```
hg.url              http://hg.intevation.org/mercurial
```

hg.tag

Optional tag which should be fetched. Can be a Mercurial tag or a revision. To prevent non-reproducible builds use of tip as revision is discouraged.

- Default: tip
- Example:

```
hg.tag              v1.3
```

```
hg.tag              ceb884843737
```

5.3.4. Checksum Phase Keywords

The list of keywords related to the checksum phase.

checksums

Checksum(s) of the distribution files. For ports with multiple distribution files, filenames must be included to associate files with their checksums.

At least two checksum types (e.g., sha1 and rmd160) should be used to ensure the integrity of the distfiles.

- Default: ???
- Examples:

```
checksums           md5     dafa161bc9c61e57636a6085c87c1fe8 \
                    sha1    5da610e1c8bc01b80abc21ab9e98e004363b429c \
                    rmd160  0c1147242adf476f5e93f4d59b553ee3ea378b23
```

```
checksums           ${distname}${extract.suffix} \
                        md5     dafa161bc9c61e57636a6085c87c1fe8 \
                        sha1    5da610e1c8bc01b80abc21ab9e98e004363b429c \
                        rmd160  0c1147242adf476f5e93f4d59b553ee3ea378b23 \
                    hobbit.tar.gz \
                        md5     3b8d02c6cf6239b9bdadbc6543c5a683 \
                        sha1    27874638b23e66d39ed94fe716ca25c967f6e993 \
                        rmd160  82b9991f3bf0ceedbf74c188c5fa44b98b5e40c9
```

5.3.5. Extract Phase Keywords

The list of keywords related to the extract phase.

extract.asroot

This keyword is used to specify that the extract phase should be done as the root user.

- Default: no
- Example:

```
extract.asroot      no
```

extract.suffix

This keyword is used to specify the extract suffix type.

- Default: .tar.gz
- Example:

```
extract.suffix      .tgz
```

use_7z

This keyword is for downloads that are compressed using the 7z algorithm. When invoked, it automatically sets:

```
extract.suffix = .7z
extract.cmd    = 7za
```

- Default: no
- Example:

```
use_7z          yes
```

use_bzip2

This keyword is for downloads that are tarred and bzipped. When invoked, it automatically sets:

```
extract.suffix = .tar.bz2
extract.cmd    = bzip
```

- Default: no
- Example:

```
use_bzip2           yes
```

use_lzma

This keyword is for downloads that are compressed using the lzma algorithm. When invoked, it automatically sets:

```
extract.suffix    = .lzma
extract.cmd       = lzma
```

- Default: no
- Example:

```
use_lzma            yes
```

use_zip

This keyword is for downloads that are zipped. When invoked, it automatically sets:

```
extract.suffix    = .zip
extract.cmd       = unzip
extract.pre_args  = -q
extract.post_args = "-d ${portpath}/${workdir}"
```

- Default: `no`
- Example:

```
use_zip             yes
```

use_xz

This keyword is for downloads that are compressed using the xz tool. When invoked, it automatically sets:

```
extract.suffix    = .tar.xz
extract.cmd       = xz
```

- Default: `no`
- Example:

```
use_xz              yes
```

extract.mkdir

This keyword is used to specify if the directory `worksrcdir` is part of the distfile or if it should be created automatically and the distfiles should be extracted there instead. This is useful for distfiles with a flat structure which would pollute the `worksrcdir` with lots of files.

- Default: `no`
- Example:

```
extract.mkdir       yes
```

extract.only, extract.only-append, extract.only-delete

List of files to extract into `${worksrcpath}`. Only use if default extract behavior is not correct for your port.

- Default: `${distfiles}`
- Example:

```
extract.only        foo.tar.gz
```

```
extract.only-append     bar.tar.gz
extract.only-delete     foo.tar.gz
```

extract.cmd

Command to perform extraction.

- Default: **gzip**
- Example:

```
extract.cmd         gunzip
```

extract.args, extract.pre_args, extract.post_args

Main arguments to `extract.cmd`; additional arguments passed before and after the main arguments.

- Default: `${distpath}/${distfile}`
- Example:

```
extract.args        ${distpath}/${distfile}
```

The following argument modifiers are available:

- extract.pre_args, defaults to: -dc
- extract.post_args, defaults to: "| tar -xf -"
- Examples:

```
extract.pre_args    xf
extract.post_args   "| gnutar -x"
```

5.3.6. Patch Phase Keywords

The list of keywords related to the patch phase.

patch.dir

Specify the base path for patch files.

- Default: ${worksrcpath}
- Example:

```
patch.dir           ${worksrcpath}/util
```

patch.cmd

Specify the command to be used for patching files.

- Default: **patch**
- Example:

```
patch.cmd           cat
```

patchfiles, patchfiles-append, patchfiles-delete

Specify patch files to be applied for a port; list modifiers specify patchfiles to be added or removed from a previous patchfile declaration.

- Default: none
- Example:

```
patchfiles          patch-Makefile.in \
                    patch-source.c
```

```
patchfiles-append   patch-configure
patchfiles-delete   patch-src-Makefile.in
```

patch.args, patch.pre_args, patch.post_args

Main arguments to patch.cmd; optional argument modifiers pass arguments before and after the main arguments.

- Default: none
- Example:

```
patch.args          ???
```

The following argument modifiers are available:

- patch.pre_args, defaults to: -p0
- patch.post_args, defaults to: none
- Examples:

```
patch.pre_args      -p1
patch.post_args     ???
```

5.3.7. Configure Phase Keywords

The list of keywords related to the configure phase.

MacPorts base sets some important default configure options, so should use the -append version of most configure keywords so you don't overwrite them. For example, MacPorts base sets default configure.cflags so you should always

use `configure.cflags-append` to set additional CFLAGS in Portfiles.

use_configure

Sets if the configure phase should be run. Can be used if the port has no `./configure` script.

- Default: `yes`
- Example:

```
use_configure    no
```

configure.env, configure.env-append, configure.env-delete

Set environment variables for configure; list modifiers add and delete items from a previous Portfile configure.env keyword, or a default set by MacPorts base. If available, it is encouraged to use the predefined options (like configure.cflags) instead of modifying configure.env directly.

- Default: `CFLAGS=-I${prefix}/include LDFLAGS=-L${prefix}/lib`
- Example:

```
configure.env       QTDIR=${prefix}/lib/qt3
```

```
configure.env-append    ABI=32
configure.env-delete    TCLROOT=${prefix}
```

configure.optflags, configure.optflags-append, configure.optflags-delete

Set optimization compiler flags; list modifiers add or delete items from a previous Portfile configure.optflags keyword or the default set by MacPorts base.

- Default: `-O2`
- Example:

```
configure.optflags    -Os
```

```
configure.optflags-append     -finline-functions
configure.optflags-delete     -O2
```

configure.cflags, configure.cflags-append, configure.cflags-delete

Set CFLAGS compiler flags; list modifiers add or delete items from a previous Portfile configure.cflags keyword or the default set by MacPorts base.

- Default: `${configure.optflags}`
- Example:

```
configure.cflags    -Os -flat_namespace
```

```
configure.cflags-append     "-undefined suppress"
configure.cflags-delete     -O2
```

configure.ldflags, configure.ldflags-append, configure.ldflags-delete

Set LDFLAGS compiler flags; list modifiers add or delete items from a previous Portfile configure.ldflags keyword or the default set by MacPorts base.

- Default: `-L${prefix}/lib`
- Example:

```
configure.ldflags   "-L${worksrcpath}/zlib -lz"
```

```
configure.ldflags-append    "-L/usr/X11R6/lib -L${worksrcpath}/lib"
configure.ldflags-delete    -L${prefix}/lib/db44
```

configure.cppflags, configure.cppflags-append, configure.cppflags-delete

Set CPPFLAGS to be passed to the C processor; list modifiers add or delete items from a previous Portfile configure.cppflags keyword or the default set by MacPorts base.

- Default: `-I${prefix}/include`
- Example:

```
configure.cppflags  -I${worksrcpath}/include
```

```
configure.cppflags-append   "-I/usr/X11R6/lib -I${worksrcpath}/lib -DHAVE_RRD_12X"
configure.cppflags-delete   -I${prefix}/lib/db44
```

configure.cxxflags, configure.cxxflags-append, configure.cxxflags-delete

Set CXXFLAGS to be passed to the C++ processor; list modifiers add or delete items from a previous Portfile configure.cxxflags keyword or the default set by MacPorts base.

- Default: `${configure.optflags}`
- Example:

```
TODO: add example
```

configure.objcflags, configure.objcflags-append, configure.objcflags-delete

TODO: add description

- Default: `${configure.optflags}`
- Example:

```
TODO: add example
```

configure.classpath, configure.classpath-append, configure.classpath-delete

TODO: add description

- Default: ???
- Example:

```
TODO: add example
```

configure.macosx_deployment_target, configure.macosx_deployment_target-append, configure.macosx_deployment_target-delete

TODO: add description

- Default: ???
- Example:

```
TODO: add example
```

configure.fflags, configure.fflags-append, configure.fflags-delete

Set FFLAGS to be passed to the Fortran compiler; list modifiers add or delete items from a previous Portfile configure.fflags keyword or the default set by MacPorts base.

- Default: `${configure.optflags}`
- Example:

```
configure.fflags    -Os
```

configure.fcflags, configure.fcflags-append, configure.fcflags-delete

Set FCFLAGS to be passed to the Fortran compiler; list modifiers add or delete items from a previous Portfile configure.fcflags keyword or the default set by MacPorts base.

- Default: `${configure.optflags}`
- Example:

```
configure.fcflags   -Os
```

configure.f90flags, configure.f90flags-append, configure.f90flags-delete

Set F90FLAGS to be passed to the Fortran 90 compiler; list modifiers add or delete items from a previous Portfile configure.f90flags keyword or the default set by MacPorts base.

- Default: `${configure.optflags}`
- Example:

```
configure.f90flags  -Os
```

configure.cc

Set CC compiler flags for selecting a C compiler.

- Default: ???
- Example:

```
configure.cc        ${prefix}/bin/gcc-mp-4.2
```

configure.cpp

Set CPP compiler flags for selecting a C preprocessor.

- Default: ???
- Example:

```
configure.cpp       /usr/bin/cpp-3.3
```

configure.cxx

Set CXX compiler flags for selecting a C++ compiler.

- Default: ???
- Example:

```
configure.cxx       /usr/bin/g++-4.0
```

configure.objc

Set OBJC compiler flags for selecting an Objective-C compiler.

- Default: ???
- Example:

```
configure.objc      /usr/bin/gcc-4.0
```

configure.fc

Set FC compiler flags for selecting a Fortran compiler.

- Default: ???
- Example:

```
configure.fc        ${prefix}/bin/gfortran-mp-4.2
```

configure.f77

Set F77 compiler flags for selecting a Fortran 77 compiler.

- Default: ???
- Example:

```
configure.f77       ${prefix}/bin/gfortran-mp-4.2
```

configure.f90

Set F90 compiler flags for selecting a Fortran 90 compiler.

- Default: ???
- Example:

```
configure.f90       ${prefix}/bin/gfortran-mp-4.2
```

configure.javac

Set JAVAC compiler flags for selecting a Java compiler.

- Default: ???

- Example:

```
configure.javac     ${prefix}/bin/jikes
```

configure.compiler

Select a compiler suite to fill the compiler environment variables. All variables/tools a compiler suite can provide are set. Manually set variables are not overwritten. Dependencies are not added for you, as they may be just build- or also run-dependencies. Keep in mind that not all compiler suites might be available on your platform: `gcc-3.3` is available on Mac OS X 10.3 and 10.4 PowerPC, `gcc-4.0` is available on 10.4+, `gcc-4.2` and `llvm-gcc-4.2` are available on 10.5 and 10.6, and clang is available on 10.6.

Only use it if a port really needs a different compiler.

- Default: `gcc-4.0` on Mac OS X 10.4 and 10.5
- Default: `gcc-4.2` with Xcode 3.2 on Mac OS X 10.6
- Default: `llvm-gcc-4.2` with Xcode 4.0 and 4.1 on Mac OS X 10.6 and 10.7
- Default: `clang` with Xcode 4.2 on Mac OS X 10.6 and 10.7
- Values: `gcc-3.3` `gcc-4.0` `gcc-4.2` `llvm-gcc-4.2` `clang` `apple-gcc-4.0` `apple-gcc-4.2` `macports-gcc-4.1` `macports-gcc-4.2` `macports-gcc-4.3` `macports-gcc-4.4` `macports-gcc-4.5`
- Example:

```
configure.compiler  macports-gcc-4.2
```

configure.perl

Set PERL flag for selecting a Perl interpreter.

- Default: `???`
- Example:

```
configure.perl      ${prefix}/bin/perl5.8
```

configure.python

Set PYTHON flag for selecting a Python interpreter.

- Default: `???`
- Example:

```
configure.python    ${prefix}/bin/python3.0
```

configure.ruby

Set RUBY flag for selecting a Ruby interpreter.

- Default: `???`
- Example:

```
configure.ruby      ${prefix}/bin/ruby
```

configure.install

Set `INSTALL` flag for selecting an install tool; used for copying files and creating directories.

- Default: `/usr/bin/install`
- Example:

```
configure.install   ${prefix}/bin/ginstall
```

configure.awk

Set AWK flag for selecting an awk executable.

- Default: `???`
- Example:

```
configure.awk       ${prefix}/bin/gawk
```

configure.bison

Set BISON flag for selecting a bison executable, a parser generator.

- Default: ???
- Example:

```
configure.bison     /usr/bin/bison
```

configure.pkg_config

Set PKG_CONFIG flag for helping find pkg_config, a tool for retrieving information about installed libraries.

- Default: ???
- Example:

```
configure.pkg_config    ${prefix}/bin/pkg-config
```

configure.pkg_config_path

Set PKG_CONFIG_PATH flag for telling pkg_config where to search for information about installed libraries.

- Default: ${prefix}/lib/pkgconfig:${prefix}/share/pkgconfig
- Example:

```
configure.pkg_config_path   ${python.prefix}/lib/pkgconfig
```

configure.args, configure.pre_args, configure.post_args

Main arguments to configure.cmd; optional argument modifiers pass arguments before and after the main arguments.

- Default: none
- Example:

```
configure.args      --bindir=${prefix}/bin
```

The following argument modifiers are available:

- configure.pre_args, defaults to: --prefix=${prefix}
- configure.post_args, defaults to: none
- Examples:

```
configure.pre_args  --prefix=${prefix}/share/bro
configure.post_args OPT="-D__DARWIN_UNIX03"
```

5.3.7.1. Configure Universal

Universal keywords are used to make a port compile on the Mac OS X platform to run on both PPC and Intel processors.

Note

There is a default universal variant made available to all ports by MacPorts base, so redefining universal keywords should only be done to make a given port compile if the default options fail to do so.

configure.universal_args

Arguments used in the configure script to build the port universal.

- Default: --disable-dependency-tracking
- Example:

```
TODO: add example
```

configure.universal_cflags

Additional flags to put in the CFLAGS environment variable when invoking the configure script. Default value is based on ${configure.universal_archs}.

- Default:

  (PowerPC Tiger) `-isysroot ${developer_dir}/SDKs/MacOSX10.4u.sdk -arch i386 -arch ppc`

  (Leopard / Intel Tiger) `-arch i386 -arch ppc`

  (Snow Leopard) `-arch x86_64 -arch i386`

- Example:

```
TODO: add example
```

configure.universal_cppflags

Additional flags to put in the CPPFLAGS environment variable when invoking the configure script.

- Default:

  (PowerPC Tiger) `-isysroot ${developer_dir}/SDKs/MacOSX10.4u.sdk`

  (Others) none

- Example:

```
TODO: add example
```

configure.universal_cxxflags

Additional flags to put in the CXXFLAGS environment variable when invoking the configure script. Default value is based on `${configure.universal_archs}`.

- Default:

  (PowerPC Tiger) `-isysroot ${developer_dir}/SDKs/MacOSX10.4u.sdk -arch i386 -arch ppc`

  (Leopard / Intel Tiger) `-arch i386 -arch ppc`

  (Snow Leopard) `-arch x86_64 -arch i386`

- Example:

```
TODO: add example
```

configure.universal_ldflags

Additional flags to put in the LDFLAGS environment variable when invoking the configure script.

- Default:

  (PowerPC Tiger) `-Wl,-syslibroot,${developer_dir}/SDKs/MacOSX10.4u.sdk -arch i386 -arch ppc`

  (Leopard / Intel Tiger) `-arch i386 -arch ppc`

  (Snow Leopard) `-arch x86_64 -arch i386`

- Example:

```
TODO: add example
```

5.3.7.2. Automake, Autoconf, and Autoreconf

The list of configure keywords available for ports that need automake and/or autoconf.

use_autoreconf

Whether or not to use autoreconf

- Default: `no`
- Example:

```
use_autoreconf      yes
```

use_automake

Whether or not to use automake.

- Default: no
- Example:

```
use_automake        yes
```

automake.env

Environment variables to pass to automake.

- Default: ???
- Example:

```
automake.env        CFLAGS=-I${prefix}/include
```

automake.args

Arguments to pass to automake.

- Default: ???
- Example:

```
automake.args       --foreign
```

automake.dir

Directory in which to run ${automake.cmd}.

- Default: ${worksrcpath}
- Example:

```
automake.dir        ./src
```

use_autoconf

Whether or not to use autoconf.

- Default: no
- Example:

```
use_autoconf        yes
```

autoconf.env

Environmental variables to pass to autoconf.

- Default: ???
- Example:

```
autoconf.env        CFLAGS=-I${prefix}/include/gtk12
```

autoconf.args

Arguments to pass to autoconf.

- Default: ???
- Example:

```
autoconf.args       "-l src/aclocaldir"
```

autoconf.dir

Directory in which to run ${autoconf.cmd}.

- Default: ${worksrcpath}
- Example:

```
autoconf.dir        src
```

5.3.8. Build Phase Keywords

The list of keywords related to the build phase.

build.cmd

Make command to run in `${worksrcdir}`. Only use it if you can't use `build.type`.

- Default: **make**
- Example:

```
build.cmd           scons
```

build.type

Defines which "make" is required. Sets `${build.cmd}` to either "gnumake" or "bsdmake" accordingly.

- Default: `gnu`
- Values: `gnu bsd pbx`
- Example:

```
build.type          bsd
```

build.args, build.pre_args, build.post_args

Main arguments to `${build.cmd}`; optional argument modifiers pass arguments before and after the main arguments.

- Default: none
- Example:

```
build.args          -DNOWARN
```

The following argument modifiers are available:

- `build.pre_args`, defaults to: `${build.target}`
- `build.post_args`, defaults to: none
- Examples:

```
build.pre_args      -project AudioSlicer.xcode
build.post_args     CFLAGS_SYS="-DUSE_FREETYPE -DPREFER_FREETYPE"
```

build.target, build.target-append, build.target-delete

Build target to pass to `${build.cmd}`; list modifiers add or delete items from a previous Portfile build.target keyword or the default set by MacPorts base.

- Default: `all`
- Example:

```
build.target        all-src
```

```
build.target-append     doc extra
build.target-delete     compat
```

build.env, build.env-append, build.env-delete

Set environment variables for build; list modifiers add and delete items from a previous Portfile build.env keyword, or a default set by MacPorts base.

- Default: none

use_parallel_build

This keyword is for specifying whether or not it is safe for a port to use multiple CPUs or multiple cores in parallel during its build phase. If `use_parallel_build` is not set to "no" in a given port, the option `-j${build.jobs}` is passed to `${build.cmd}` (if `${build.cmd}` is **make** or **scons**).

- Default: `yes`
- Example:

- Example:

```
use_parallel_build  no
```

build.jobs

The number of simultaneous jobs to run when parallel build is enabled. The default value is based on the variable `buildmakejobs` in `macports.conf`.

- Default: If `buildmakejobs` is 0, the number of CPU cores in the machine, or the number of GB of physical memory plus one, whichever is less. Otherwise, the actual value of `${buildmakejobs}`.

5.3.9. Test Phase Keywords

The list of keywords related to the test phase.

test.run

Enable running test suites bundled with a port.

- Default: `no`
- Example:

```
test.run            yes
```

test.cmd

Test command to run relative to `${worksrcdir}`.

- Default: `${build.cmd}`
- Example:

```
test.cmd            checks.sh
```

test.target

Test target to pass to `${test.cmd}`.

- Default: `test`
- Example:

```
test.target         checks
```

test.env, test.env-append, test.env-delete

Set environment variables for test; list modifiers add and delete items from a previous Portfile test.env keyword, or a default set by MacPorts base.

Often `DYLD_LIBRARY_PATH` is set here to support testing dynamically linked libraries.

- Default: none
- Example:

```
test.env        DYLD_LIBRARY_PATH=${worksrcpath}/src/.libs
```

5.3.10. Destroot Phase Keywords

The list of keywords related to the destroot phase.

destroot.cmd

Install command to run relative to `${worksrcdir}`.

- Default: `${build.cmd}`
- Example:

```
destroot.cmd        scons
```

destroot.args, destroot.pre_args, destroot.post_args

Main arguments to `${destroot.cmd}`; optional argument modifiers pass arguments before and after the main arguments.

- Default: none
- Example:

```
destroot.args       BINDIR=${prefix}/bin
```

The following argument modifiers are available:

- `destroot.pre_args`, defaults to: `${destroot.target}`
- `destroot.post_args`, defaults to: none
- Examples:

```
destroot.pre_args   -project AudioSlicer.xcode
destroot.post_args  INSTDIR=${destroot}${prefix}
```

destroot.target, destroot.target-append, destroot.target-delete

Install target to pass to `${destroot.cmd}`; list modifiers add or delete items from a previous Portfile destroot.target keyword or the default set by MacPorts base.

- Default: `install`
- Example:

```
destroot.target     install install-config install-commandmode
```

```
destroot.target-append  install-plugins
destroot.target-delete  install-commandmode
```

destroot.destdir

Arguments passed to `${destroot.cmd}` to install correctly into the destroot.

- Default: `DESTDIR=${destroot}`
- Example:

```
destroot.destdir    prefix=${destroot}${prefix}
```

Note

If an application's Makefile properly supports the DESTDIR variable, MacPorts will automatically destroot the port properly. A port must destroot properly or the port will not install correctly, upgrade, or uninstall. If not, you may need to set this variable, or even patch the application's Makefile.

destroot.umask

Umask to use during destroot.

- Default: `022`
- Example:

```
destroot.umask      002
```

destroot.keepdirs

A list of directories that should not be removed if empty upon destroot completion.

- Default: ???
- Example:

```
destroot.keepdirs   ${destroot}${prefix}/var/run \
                    ${destroot}${prefix}/var/log \
                    ${destroot}${prefix}/var/cache/mrtg
```

destroot.violate_mtree

MacPorts tests for compliance to the common directory structure in `${prefix}`. If a port is not compliant with the standard, set it to `yes`.

You can find the macports standard in MacPorts File Hierarchy or in the porthier(7) man page.

If `destroot.violate_mtree` is set to `yes`, the following warning is issued during the installation.

```
Warning: portname requests to install files outside the common directory structure!
```

This means that the port installed files outside of their normal locations in `${prefix}`. These could be files totally outside of `${prefix}`, which could cause problems on your computer, or files inside of `${prefix}` that are not in a standard location. Use `port contents portname` to see the location for all files that were installed by a given port.

- Default: `no`
- Example:

```
destroot.violate_mtree      yes
```

5.4. Dependencies

Free and open source software is highly modular, and MacPorts ports often require that other ports be installed beforehand; these prerequisites for a given port are called a port's "dependencies".

The keywords used when specifying dependencies in a Portfile are related to port install phases, and they refer to what are called library, build, fetch, extract and run dependencies. Though all of them install dependencies before a given port is installed, specifying dependencies with the correct keyword is important for proper port upgrade and uninstall behavior, or when running targets other than install. For example, you may not uninstall a port that is a library dependency for another installed port, though you may remove one that is a build dependency. Likewise, if you run the fetch target for a port, only the fetch dependencies will be installed first, so they should be all that is needed for that target.

depends_fetch, depends_fetch-append, depends_fetch-delete

The list of dependencies to check before phases `fetch`, `checksum`, `extract`, `patch`, `configure`, `build`, `destroot`, `install`, and `package`. Fetch dependencies are needed to download the distfiles for a port, and are not needed at all once the software is installed.

depends_extract, depends_extract-append, depends_extract-delete

The list of dependencies to check before phases `extract`, `patch`, `configure`, `build`, `destroot`, `install`, and `package`. Extract dependencies are needed to unpack a port's distfiles into the work directory, and are not needed at all once the software is installed.

depends_build, depends_build-append, depends_build-delete

The list of dependencies to check before phases `configure`, `build`, `destroot`, `install`, and `package`. Build dependencies are needed when software is being built, but not needed at all once it is installed.

depends_lib, depends_lib-append, depends_lib-delete

The list of dependencies to check before phases `configure`, `build`, `destroot`, `install`, and `package`. Library dependencies are needed both at build time (for headers and libraries to link against) and at run time.

depends_run, depends_run-append, depends_run-delete

The list of dependencies to check before phases `destroot`, `install`, and `package`. Run dependencies are needed when the software is run, but not to compile it.

5.4.1. Port and File Dependencies

There are two types of dependencies: port dependencies and file dependencies. Port dependencies can be satisfied by reference to a port (the MacPorts registry is queried), or by reference to a file (whether provided by a port or not). The most commonly-used type of dependencies in Portfiles are port dependencies, because dependencies should be provided by MacPorts ported software whenever possible, and usually only one port can provide the needed libraries and files.

But when satisfying a dependency with vendor-supplied software is preferred for special reasons, or when it is possible for more than one port to satisfy a dependency, then file dependencies may be used. An example of the former is with X11 —Apple's is widely preferred; an example of the latter is with "-devel" named ports —these ports provide identical files (though only one can be activated at a time).

Port dependencies, the preferred type, are specified as shown in these examples:

```
depends_lib         port:rrdtool port:apache2

depends_build       port:libtool

depends_run         port:apache2 port:php5
```

File dependencies may be of three types: `bin` for programs, `lib` for libraries, and `path` for any installed file. File dependencies are specifed in the form: *<type>*:*<filespec>*:*<port>*.

For `bin` dependencies, *<filespec>* is the name of a program in a bin directory like `${prefix}/bin`, /usr/bin, /bin, and the associated sbin directories.

For `lib` dependencies, *<filespec>* is the name of a library (but without its extension) in a lib directory like `${prefix}/lib`, /usr/lib, /lib, some Framework directories, and those found in environment variables like DYLD_LIBRARY_PATH.

For `path` dependencies, *<filespec>* is the complete absolute path to the file, or more usually, when the file is inside `${prefix}`, it is specified relative to `${prefix}`. Since `path` dependencies are the only ones which would find files only in an absolute path or a path inside `${prefix}` they are - in cases when a port needs to be more restrictive - often used instead of `bin` and `lib` dependencies .

Note that the *<port>* specified is only installed if the specified library, binary, or file is not found. See the examples below:

```
depends_lib         lib:libX11.6:xorg

depends_build       bin:glibtool:libtool

depends_run         path:lib/libltdl.a:libtool
```

5.5. Variants

MacPorts variants are conditional modifications of port installation behavior during port installation. There are two types of variants: user-selected variants and platform variants. User-selected variants are options selected by a user when a port is installed; platform variants are selected automatically by MacPorts base according to the OS or hardware platform (darwin, freebsd, linux, i386, powerpc, etc.).

5.5.1. User-Selected Variants

User-selected variants are those that are defined so a user can invoke them to enable port options at install time. They also allow a port author a level of modularity and control using the keyword `default_variants` (see below).

> **Note**
>
> Variant names may contain only letters, numbers and underscore characters. In particular, the hyphen is not a valid character in variant names because it would conflict with the notation for deselecting a variant.

variant *name* [requires *variant1 variant2 ...*] [conflicts *variant1 variant2 ...*] [description *description*]

The variant declaration may contain any keywords that can be placed in a Portfile's global section. If you wish to execute system (shell) calls or Tcl extensions during the execution of a port phase, you should place those statements within a `variant_isset` conditional within a phase declaration and not within the variant declaration itself. Dependencies and conflicts with other variants in the same port can be expressed with "requires" and "conflicts" options as shown below.

- Default: none
- Examples:

```
variant gnome requires glib {
    configure.args-append   --with-gnome
    depends_lib-append      port:gnome-session
}
```

```
variant apache2 conflicts apache {
    configure.args-append \
        --with-apxs2=${prefix}/apache2/bin/apxs
}
```

default_variants

The optional `default_variants` keyword is used to specify variants that a port author wishes to have enabled by default. This allows for Portfile modularity and also allows users to suppress default variants if they wish.

- Default: none
- Example:

```
default_variants    +ssl +tcpd
```

Default variants may be suppressed by preceding a variant name with a "-" as shown in this example.

```
%% port install foo -ssl
```

universal_variant

When using MacPorts on Mac OS X, a universal variant is defined by default to configure ports with universal flags. The variant can be overridden if the default code does not work (see the Configure Universal section above), or suppressed if a universal variant does not function properly for a given port.

- Default: `yes`
- Example:

```
universal_variant   no
```

5.5.2. User-Selected Variant Descriptions

User-selected variants ought to provide a description, which will be displayed when using command "port variants foo". The syntax used for the description keyword is shown below.

```
variant bar description {Add IMAP support} {}
```

Descriptions should be short but clear, and not merely repeat the name of the variant. To allow for compatibility for possible MacPorts GUI support, a good rule of thumb is to use sentence fragments for brevity, with a capitalized first letter and no trailing punctuation. Think of them as short labels such as ones you'd find next to a GUI checkbox or radio button. Thus, it would be better to write "Build with support for foo" instead of "Builds with support for foo"; "Add support for foo" would be better than "Adds support for foo".

Variant descriptions are strings, so one should take care not to put whitespace between the brackets and the beginning and end of the variant description, and also not to use unnceccessary whitespace, unlike with port descriptions and long_descriptions.

5.5.3. Platform Variants

Platform variants are either defined by default in MacPorts base, or defined by a port author to customize a port's installation according to OS (operating system) or hardware platform.

platform *os* [*version*] [*arch*]

MacPorts allows platform-specific port options to be specified in a Portfile for handling differences between platforms and versions of the same platform.

`platform darwin` *version* can be used to handle different tasks depending on the version of Darwin, the core operating system underlying Mac OS X. *version* is the major version of Darwin, and can be `8` for Mac OS X 10.4 Tiger, `9` for 10.5 Leopard, `10` for 10.6 Snow Leopard or `11` for 10.7 Lion.

- Examples:

```
platform darwin 10 {
    configure.env-append LIBS=-lresolv
}
```

```
platform darwin i386 {
    configure.args-append --disable-mmx
}
```

```
platform darwin 8 powerpc {
    configure.compiler gcc-3.3
}
```

Note

Though a combination of OS version and hardware platform may be specified in a single platform statement (i.e. darwin 8 i386), it is not possible to specify a range of platforms with a single statement. For example, to select Darwin versions 9 and 10 while excluding all others, you would need two statements: `platform darwin 9` and `platform darwin 10`. Alternately, you could make that behavior the port's default, and add a `platform darwin 8` block to remove it again.

5.6. Tcl Extensions

A MacPorts Portfile is a Tcl script, so it may contain any arbitrary Tcl code you may learn about in a Tcl reference manual. However, few authors will use arbitrary Tcl code; the vast majority will use Tcl extensions that are coded within MacPorts for performing the most common tasks needed for Portfiles. The list below is a list of Tcl extensions provided by MacPorts base.

file

Description.

file copy

file rename

file delete [-force]

file mkdir

macros

Description.

copy

Shorthand alternative to "file copy".

move

Shorthand alternative to "file rename".

delete file ...

Deletes each of the given files/directories. Behaves similarly to file delete -force except that file delete -force will fail to delete directories properly on 10.3 systems.

touch

Mimicks the BSD touch command.

ln

Mimickes the BSD ln command.

xinstall

xinstall copies files and creates directories; it is intended to be compatible with install(1).

xinstall [-o *owner*] [-g *group*] [-m *mode*] [*file1 file2 ...*] *directory*

Install the specified file(s) to a destination directory.

xinstall [-o *owner*] [-g *group*] [-m *mode*] [-W *dir*] [*file1 file2 ...*] *directory*

Change to `dir` and install file(s) to a destination directory.

eval xinstall [-o *owner*] [-g *group*] [-m *mode*] [glob *pattern*] *directory*

Install the file(s) matching the glob pattern to a destination directory.

xinstall -d [-o *owner*] [-g *group*] [-m *mode*] *directory*

Create a directory including parent directories if necessary.

Defaults:

- owner -
- group -
- mode -

Examples:

```
xinstall -m 640 ${worksrcpath}/doc README \
   ${destroot}${prefix}/share/doc/${name}
```

```
xinstall -m 640 -W ${worksrcpath}/doc README INSTALL COPY \
   ${destroot}${prefix}/share/doc/${name}
```

```
eval xinstall -m 640 [glob ${worksrcpath}/doc/*] \
   ${destroot}${prefix}/share/doc/${name}
```

```
xinstall -d ${destroot}${prefix}/share/doc/${name}
```

strsed

strsed can be used for string manipulations using regular expressions. It supports a small subset of the commands known from sed(1).

strsed *string* **s/***regex***/***replacement***/**

Replaces the first instance of *regex* with *replacement*. Refer to re_format(7) for a definition of regular expression syntax.

strsed *string* **g/***regex***/***replacement***/**

The same as the previous format, except all instances of the pattern will be replaced, not only the first (mnemonic: 'g' is for global).

reinplace

Allows text specified by a regular expression to be replaced by new text, in-place (the file will be updated itself, no need to place output into a new file and rename).

reinplace [-E] [--] *command file ...*

Replace text given by the regular expression portion of the command with the replacement text, in all files specified.

Use -E to use the extended regular expression style (see re_format(7) for a description of the basic and extended styles)

Use -- to end option processing and allow any further dashes not to be treated as options.

Examples:

```
reinplace "s|/usr/local|${prefix}|g" ${worksrcpath}/configure
```

```
reinplace "s|@@PREFIX@@|${prefix}|g" ${worksrcpath}/Makefile
```

user/group

adduser username [uid=*uid***] [gid=***gid***] [passwd=***passwd***] [realname=***realname***] [home=***home***] [shell=***shell***]**

Add a new local user to the system with the specified uid, gid, password, real name, home directory and login shell.

existsuser *username*

Check if a local user exists. Returns the uid for the given user, or 0 if the user wasn't found. Checking for the root user is not supported because its uid is 0, and it will always exist anyway.

nextuid

Returns the highest used uid plus one.

addgroup *group* **[gid=***gid***] [passwd=***passwd***] [realname=***realname***] [users=***users***]**

Add a new local group to the system, with the specified gid, password, real name, and with a list users as members.

existsgroup *group*

Check if a local group exists and return the corresponding gid. This can be used with adduser:

```
addgroup foo
adduser foo gid=[existsgroup foo]
```

nextgid

Returns the highest used gid plus one.

External program execution

Use only when ....

5.7. StartupItems

A StartupItem is a MacPorts facility to run "daemons," a Unix term for programs that run continuously in the background, rather than under the direct control of a user; for example, mail servers, network listeners, etc. Ports that use StartupItem keywords create Mac OS X scripts for launchd, which is the Apple facility introduced with Mac OS X 10.4 to replace xinetd for starting and managing daemons. To support **launchd**, a program named **daemondo** is provided by MacPorts base that serves as an adapter between Mac OS X's **launchd** and daemons ("executable"

StartupItems) or traditional Unix startup scripts that start daemons ("script" StartupItems).

There are three categories of StartupItem keywords. Those that trigger StartupItem creation and logging, those that specify attributes of "executable" StartupItems, and those that specify attributes of "script" StartupItems.

Note

The variable `startupitem_type` in `${prefix}/etc/macports/macports.conf` may be set to `none` to globally override all StartupItem keywords found in Portfiles; this prevents StartupItems from being created.

5.7.1. StartupItem Attributes

The keywords in this section may be used with either "executable" or "script" StartupItems (see below).

startupitem.create

Trigger the creation of a StartupItem.

- Default: `no`
- Example:

```
startupitem.create      yes
```

startupitem.name

Sets the name for the StartupItem. Defaults to the name of the port, so this keyword is usually unnecessary.

- Default: `${name}`
- Example:

```
startupitem.name        dhcpd
```

startupitem.logfile

Path to a logfile for logging events about the lifetime of the StartupItem. Depending on the type of StartupItem, and the manner in which it is started, standard output from the daemon may also be directed to the logfile.

- Default: `/dev/null`
- Example:

```
startupitem.logfile     ${prefix}/var/log/mydaemon.log
```

startupitem.logevents

Control whether or not to log events to the log file. If logevents is set, events with timestamps are logged to the logfile.

- Default: `no`
- Example:

```
startupitem.logevents   yes
```

startupitem.netchange

Cause the daemon to be restarted when a change in network state is detected.

- Default: `no`
- Example:

```
startupitem.netchange   yes
```

5.7.2. Executable StartupItems

Daemons run continuously, so monitoring the health of daemon processes and restarting them if they die is an important StartupItems' feature. "Executable" StartupItems are preferred over "script" StartupItems because **daemondo** launches the daemon ***directly***, rather than ***indirectly*** via a script, and therefore it automatically knows how to monitor a daemon process and restart it if it dies. Daemons used with "executable" StartupItems may be programs or scripts (shell, perl, python, etc.) as long as the script ***itself*** is the daemon, rather than merely what launches the daemon. In the latter case "script" StartupItems are to be used.

Note

Since "script" and "executable" are mutually exclusive StartupItem types, the `startupitem.executable` keyword may not be used in a Portfile that uses any keywords listed in the Script StartupItems section.

startupitem.executable

Specifies the name of the daemon to be run. It may have multiple arguments, but they must be appropriate for a call to exec; arbitrary shell code may not be used.

Note

Some daemons "daemonize" by detaching themselves from the controlling tty before sending themselves to the background, thus making themselves a child of the original process. A daemon to be started with `startupitem.executable` must not be allowed to do this or daemondo will think the process has died and start multiple instances. Often daemons have a command switch to run in the foreground, and this method should be used for daemons that detach.

- Default: none
- Example:

```
startupitem.executable  ${prefix}/sbin/vm-pop3d -d 10 -t 600
```

Note

Do not wrap values in quotes if passing arguments to the daemon; "executable" StartupItem elements must be tagged individually so the spaces between arguments serve as delimiters for "string" tags. For example, this startupitem key/value pair:

```
startupitem.executable    ${prefix}/sbin/vm-pop3d -d 10 -t 600
```

generates a .plist file with these tags:

```
<key>ProgramArguments</key>
<array>
    <string>/opt/local/bin/daemondo</string>
    <string>--label=vm-pop3d</string>
    <string>--start-cmd</string>
    <string>/opt/local/sbin/vm-pop3d</string>
    <string>-d</string>
    <string>10</string>
    <string>-t</string>
    <string>600</string>
    <string>;</string>
</array>
```

5.7.3. Script StartupItems

StartupItems of type "script" create a wrapper during port installation for **daemondo** that that will be used to launch a daemon startup script present in an application's source distribution (MacPorts does not create daemon startup scripts) for daemons that require a script.

Note

"Executable" StartupItems are the preferred type since "script" StartupItems launch daemons ***indirectly***, and this requires that port authors use the `startupitem.pidfile` keyword so that **daemondo** can check this pid file to see is a daemon process has died and restart it. Any time a script (or an executable) itself serves as a daemon, use the "executable" StartupItem type so daemondo will launch it directly and track its health automatically. Additionally, since "script" and "executable" are mutually exclusive StartupItem types, the `startupitem.executable` keyword may not be used in a Portfile that uses "script" StartupItem keywords.

A typical snippet of a startup script that may be used with a "script" StartupItem is shown below. Notice that the script is not a daemon; rather the script indirectly launches the vm-pop3d daemon.

```
#!/bin/sh

case "$1" in
    start)
        echo -n "Starting vm-pop3d: "

        /opt/local/sbin/vm-pop3d -d 10 -t 600

[... trimmed ...]
```

startupitem.start, startupitem.stop, startupitem.restart

Specify a shell script to start, stop, and restart the daemon. In the absence of `startupitem.restart`, the daemon will be restarted by taking the stop action, followed by the start action.

- Default: none
- Examples:

```
startupitem.start       "${prefix}/share/mysql/mysql.server start"
startupitem.stop        "${prefix}/share/mysql/mysql.server stop"
startupitem.restart     "${prefix}/share/mysql/mysql.server restart"
```

Note

Wrap the stop, start, and restart values in quotes so they will be placed in the wrapper tagged as a single element.

startupitem.init

Shell code that will be executed prior to any of the options `startupitem.start`, `startupitem.stop` and `startupitem.restart`.

- Default: none
- Example:

```
startupitem.init        BIN=${prefix}/sbin/bacula-fd
```

startupitem.pidfile

This keyword must be defined properly for **daemondo** to be able to monitor daemons launched via "script" StartupItems and restart them if they die. It specifies two things: a process id (PID) file handling method, and a pidfile name and path.

- Default: `none ${prefix}/var/run/${name}.pid`

  Default: [none] | [`${prefix}/var/run/${name}.pid`]
- Values [none auto manual clean] [*`/path/to/pidfile`*]
- Example:

```
startupitem.pidfile     auto ${prefix}/var/run/${name}.pidfile
```

PID file handling options:

- `none` - daemondo will not create or track a PID file, so it won't know when a daemon dies.
- `auto` - The started process is expected to create a PID file that contains the PID of the running daemon; daemondo then reads the PID from the file and tracks the process. The started process must delete the PID file if this is necessary.
- `clean` - The started process is expected to create a PID file that contains the PID of the running daemon; daemondo then reads the PID from the file and tracks the process, and deletes the PID file if it detects the daemon has died.
- `manual` - This option should only be used if an "executable" StartupItem could be used (daemondo launches a daemon directly) ***and*** a port author wants a PID file written for some special use. A PID file is not needed to detect process death for daemons launched directly by daemondo. As with executable StartupItems, daemondo remembers the PID of the launched process and tracks it automatically.

5.7.4. Loading / Unloading StartupItems into launchd

A port with a StartupItem places a link to a .plist file for the port's daemon within `/Library/LaunchDaemons/`. A .plist file is an XML file; MacPorts installs .plist files tagged as "disabled" for the sake of security. You may enable a startup script (tag the.plist file as "enabled") and load it into **launchd** with a single command as shown.

```
%% sudo launchctl load -w /Library/LaunchDaemons/org.macports.mysql5.plist
```

You may stop a running startup script, disable it (tag the.plist file as "disabled"), and unload it from **launchd** with a single command as shown.

```
%% sudo launchctl unload -w /Library/LaunchDaemons/org.macports.mysql5.plist
```

5.7.5. StartupItem Internals

During port installation a MacPorts StartupItem creates a .plist file in `${prefix}/etc/LaunchDaemons/`, and places a

symbolic link to the .plist file within /Library/LaunchDaemons/.

For example, the StartupItem for the mysql5 port is org.macports.mysql5.plist, and it is linked as shown.

```
%% ls -l /Library/LaunchDaemons
```

```
org.macports.mysql5.plist ->
     /opt/local/etc/LaunchDaemons/org.macports.mysql5/org.macports.mysql5.plist
```

For "script" StartupItems, in addition to a .plist file, a wrapper is also created.

```
%% ls -l /opt/local/etc/LaunchDaemons/org.macports.mysql5/
```

```
-rwxr-xr-x   2 root  wheel  475 Aug  2 14:16 mysql5.wrapper
-rw-r--r--   2 root  wheel  975 Aug  2 14:16 org.macports.mysql5.plist
```

The wrapper manipulates the script as specified in the startupitem.start and startupitem.stop keywords. An example wrapper script snippet is shown below.

```
#!/bin/sh

# MacPorts generated daemondo support script

# Start
Start()
{
    /opt/local/share/mysql5/mysql/mysql.server start
}

# Stop
Stop()
{
    /opt/local/share/mysql5/mysql/mysql.server stop
}

[... trimmed ...]
```

5.8. Livecheck / Distcheck

Options livecheck and distcheck are especially useful for port maintainers, but others may also find this information valuable.

Livecheck checks to see if MacPorts can query the developer's download site to determine if a newer version of the software has become available since the port was installed.

livecheck.type

Specify what kind of update check to perform.

Open source mirror site options are to use the project's latest file release from sourceforge or googlecode, or the project's date_updated XML tag for freshmeat. These options are automatically used if a matching ${master_sites} URL is used.

Generic download site options are to specify a moddate (modification date of a URL resource), a regex (retrieve the version by applying a regex to a URL resource), regexm (retrieve the version by applying a multi-line regex to a URL resource), md5 (compares the md5 sum of a URL resource) or none (no check).

- Default: sourceforge or googlecode if the ${master_sites} is one of these, else freshmeat.
- Values: freshmeat sourceforge googlecode moddate regex regexm md5 none
- Examples:

```
livecheck.type     regex
livecheck.url       ${homepage}
livecheck.regex     "Generally Available (\\d+(?:\\.\\d+)*)"
```

livecheck.name

Name of the project for live checks. Is only used with freshmeat, sourceforge, and googlecode livechecks.

- Default: ${name} or the sourceforge, freshmeat or googlecode project name if it can be guessed from ${master_sites}.
- Example:

```
livecheck.name      hibernate
```

livecheck.distname

Name of the file release for sourceforge and googlecode checks. For sourceforge releases use the name of the package release. For googlecode releases use the name of the file download, including extension. You may use this keyword without `livecheck.version` if you replace the version part of the name with "`(.*)`".

- Default: sourceforge: `${livecheck.name}`, googlecode: first `${distfiles}` item
- Example:

```
livecheck.distname  faad2.src
```

livecheck.version

Version of the project for a check; used for regex-based checks.

- Default: `${version}`
- Example:

```
livecheck.version   ${name}-${version}
```

livecheck.url

URL to query for a check.

- Default:
  - `${homepage}` or the first hit among the following sites:
  - `http://freshmeat.net/projects-xml/${livecheck.name}/${livecheck.name}.xml`
  - `http://sourceforge.net/api/file/index/project-name/${livecheck.name}/rss`
  - `http://code.google.com/p/${livecheck.name}/downloads/list`
- Example:

```
livecheck.url       http://ftp.gnu.org/gnu/bison/
```

livecheck.regex

Regular expression to parse the resource for regex checks. Be sure to use a regular expression grouping around the version component.

- Default: none
- Example:

```
livecheck.regex     4th-([a-z0-9.]+)-unix${extract.suffix}
```

livecheck.md5

md5 checksum to use for an md5 comparison.

- Default: none
- Example:

```
livecheck.md5       37e6a5b6516a680c7178b72021d3b706
```

Distcheck reports whether or not the distfile(s) specified in a Portfile are still available on the developer's download site. Examples are given below.

distcheck.check

This option can be used to disable distcheck. It specifies what kind of check should be performed on distfiles: `moddate` (check if the Portfile is older than the distfile) or `none` (no check).

- Default: `moddate`
- Example:

```
distcheck.check     none
```

5.9. PortGroups

5.9.1. PortGroup introduction

PortGroups are simply include files for portfiles. They can define as much or as little as a portgroup author feels is necessary to provide a set of definitions or behaviors common to a group of portfiles, in order that those portfiles can be expressed as simply as possible with minimum redundancy.

See the following folder for PortGroup definitions:

${prefix}/var/macports/sources/rsync.macports.org/release/ports/_resources/port1.0/group/

A sample listing follows:

```
%% ls -1 /opt/local/var/macports/sources/rsync.macports.org/release/ports/_resources/port1.0/group/
```

```
archcheck-1.0.tcl
cmake-1.0.tcl
crossbinutils-1.0.tcl
gnustep-1.0.tcl
haskell-1.0.tcl
hocbinding-1.0.tcl
hunspelldict-1.0.tcl
kde4-1.0.tcl
kde4-1.1.tcl
.
.
.
```

The requirements of a minimum portfile using a portgroup varies by portgroup. The sections below devoted to each portgroup (or, for portgroups not documented there yet, the comments in the header of the portgroup file itself) should provide guidance on how each portgroup is used. Prospective MacPorts developers are also encouraged to examine existing portfiles that use these portgroups.

5.9.2. PortGroup gnustep

PortGroup gnustep allows for efficient porting of GNUstep-based open source software using the GNU objective-C runtime that defines options for the configuration, build, and destroot phases, and also defines some values for GNUstep-based software. A minimum Portfile using the gnustep PortGroup class need only define the fetch and the checksum phases.

5.9.2.1. gnustep PortGroup Specific Keywords

Portfiles using the gnustep PortGroup allow for port authors to set the following keywords in addition to the general Portfile keywords.

gnustep.post_flags

An associative array which specifies the sub-directories relative to ${worksrcpath} and the SHARED_LD_POSTFLAGS variables to be added to GNUmakefile.preamble in those sub-directories. This helps making the patching process easier on Darwin.

- Type: optional
- Default: none
- Example:

```
platform darwin {
    array set gnustep.post_flags {
        BundleSubDir "-lfoo -lbar"
    }
}
```

gnustep.cc

Define the gcc compiler to use when compiling a port.

- Type: optional
- Default: gcc-mp-4.2
- Example:

```
gnustep.cc gcc-mp-4.3
```

variant with_docs

Many GNUstep packages include a Documentation sub-directory that is not built by default. Enabling this variant

Many GNUstep packages include a Documentation sub-directory that is not built by default. Enabling this variant builds and installs the included documentation.

- Type: optional
- Example:

```
%% port install gnustep-gui +with_docs
```

5.9.2.2. gnustep FilesystemLayout Keywords

PortGroup gnustep supports both the traditional gnustep file layout and the new fhs file layout. However, a given ported application does not necessarily support both. The Portfiles have access to many procedures to handle these two layouts:

set_gnustep_make

Sets GNUSTEP_MAKEFILES according to the FilesystemLayout

set_gnustep_env

Sets DYLD_LIBRARY_PATH and PATH for the gnustep FilesystemLayout

gnustep_layout

Returns true (1) if current file layout is gnustep

set_system_library

Sets GNUSTEP_SYSTEM_LIBRARY according to the FilesystemLayout

set_local_library

Sets GNUSTEP_LOCAL_LIBRARY according to the FilesystemLayout

5.9.2.3. gnustep PortGroup Sugar

Portfiles using PortGroup gnustep do not need to define the following variables:

categories

Default: gnustep

homepage

Default: http://www.gnustep.org/

master_sites

Default: gnustep:core

depends_lib

Default: gnustep-core

use_configure

Default: no

configure.env

Default: DYLD_LIBRARY_PATH PATH

configure.pre_args-append

Default: CC=gcc-mp-4.2 GNUSTEP_MAKEFILES

build.type

Default: gnu

build.env

Default: DYLD_LIBRARY_PATH PATH

build.pre_args-append

Default: messages=yes

destroot.env

destroot.env

Default: DYLD_LIBRARY_PATH PATH

destroot.pre_args-append

Default: messages=yes

5.9.3. PortGroup haskell

PortGroup haskell simplifies the addition of Haskell packages.

5.9.3.1. haskell PortGroup Specific Keywords

Portfiles using the haskell PortGroup allow for port authors to set the following keywords in addition to the general Portfile keywords.

haskell.setup

This keyword sets a number of port variables.

- Type: required
- Synopsis: the first argument is the package name, as called by hackageDB; the second is the version number
- Example:

```
haskell.setup   digest 0.0.0.2
```

5.9.3.2. haskell PortGroup Sugar

Portfiles using PortGroup haskell do not need to define the following variables:

name

Default: hs-[string tolower ${package}]

version

Default: ${version} (from haskell.setup)

categories

Default: devel haskell

homepage

Default: http://hackage.haskell.org

master_sites

Default: ${homepage}/packages/archive/${package}/${version}

distname

Default: ${package}-${version}

depends_build

Default: ghc

configure, build, and destroot phases

Default: proper setup to run these phases

post-destroot

Default: creates and installs (into destroot) the register.sh and unregister.sh scripts

post-activate

Default: runs the register.sh scripts

livecheck

Default: runs livecheck against the package's information page

5.9.4. PortGroup perl5

5.9.4. PortGroup perl5

PortGroup perl5 allows for efficient porting of perl modules and other perl open source software.

5.9.4.1. perl5 PortGroup Specific Keywords

Portfiles using the perl5 PortGroup allow for port authors to set the following keywords in addition to the general Portfile keywords.

perl5.setup

This keyword sets the ${distfile} and ${version}.

- Type: required
- Example:

```
perl5.setup           Net-Telnet 3.03
```

5.9.4.2. perl5 PortGroup Sugar

Portfiles using PortGroup perl5 do not need to define the following variables:

categories

Default: perl

master_sites

Default: http://search.cpan.org/dist/${distname}

depends_lib

Default: perl5.8

use_configure

Default: no

5.9.4.3. perl5 PortGroup Specific Variables

When the perl5 PortGroup is declared within a Portfile, the following variables are provided during port install.

perl5.version

The MacPorts Perl version.

perl5.bin

The Perl binary path (i.e., `${prefix}/bin/perl`).

perl5.lib

Path to the Perl vendor directory.

perl5.archlib

Path to the Perl architecture-dependent modules directory.

5.9.5. PortGroup python & python2x

PortGroup python allows for efficient porting of python-based open source software.

5.9.5.1. python PortGroup Specific Keywords

Portfiles using the python2x PortGroup allow for port authors to set the following keywords in addition to the general Portfile keywords.

python.link_binaries

When yes (the default), tells the PortGroup to automatically link any executable binaries installed in the bin/ directory within the framework into `${prefix}/bin`.

- Type: optional
- Example:

```
python.link_binaries      no
```

python.add_archflags

When yes (the default), the PortGroup will automatically try to pass the correct arch-specific flags during build time (via the standard CFLAGS, LDFLAGS, etc environment variables). Set this to no and setup those variables in build.env manually if the default does not work.

- Type: optional
- Example:

```
python.add_archflags      no
```

5.9.5.2. python PortGroup Specific Variables

When the python PortGroup is declared within a Portfile, the following variables are provided during port install.

python.bin

The MacPorts Python binary location.

python.lib

The Python dynamic library and path (i.e., `${prefix}/lib/libpython2.x.dylib`).

python.include

Path to the Python include directory.

python.pkgd

Path to the Python site-packages directory. (i.e., `${prefix}/lib/python2.4/site-packages`).

5.9.5.3. python PortGroup Sugar

Portfiles using PortGroup python do not need to define the following variables:

categories

Default: python

dist_subdir

Default: python

depends_lib

Default: port:pythonXY (XY as appropriate for the group)

use_configure

Default: no

build.cmd

Default: ${python.bin} setup.py --no-user-cfg

build.target

Default: build

destroot.cmd

Default: ${python.bin} setup.py --no-user-cfg

destroot.destdir

Default: --prefix=${prefix} --root=${destroot}

pre-destroot

Default: creates ${destroot}${prefix}/share/doc/${name}/examples

5.9.6. PortGroup ruby

PortGroup ruby allows for efficient porting of ruby-based open source software.

5.9.6.1. ruby PortGroup Specific Variables

When the ruby PortGroup is declared within a Portfile, the following variables are provided during port install.

ruby.version

The MacPorts Ruby version.

ruby.bin

The Ruby binary location.

ruby.lib

Path to the Ruby vendorlibdir directory (i.e., `${prefix}/lib/ruby/vendor_ruby/${ruby.version}`)

ruby.arch

The name for the Ruby architecture-dependent directory name (i.e., `i686-darwin8.10.1`).

ruby.archlib

Path to the Ruby vendor archdir (i.e., `${ruby.lib}/${ruby.arch}`).

5.9.7. PortGroup xcode

`PortGroup xcode` allows for efficient porting of Xcode-based opensource software. A minimum Portfile for `PortGroup xcode` uses defaults for the configuration, build, and destroot phases. It also defines some values for Xcode-based software.

Using `PortGroup xcode` is a way to make your port able to tolerate Xcode version updates because the PortGroup is tested against all supported Mac OS X and Xcode versions.

5.9.7.1. xcode PortGroup Specific Keywords

Portfiles using `PortGroup xcode` allow for port authors to set the following keywords in addition to the general Portfile keywords.

xcode.project

The path relative to `${build.dir}` and `${destroot.dir}` of the Xcode project. If unset, let Xcode Tools should determine it automatically. It usually succeeds if there is a single project in the directory.

- Type: optional
- Default: none
- Example:

```
xcode.project ${name}.xcode
```

xcode.configuration

Project configuration/buildstyle to use.

- Type: optional
- Default: `Deployment`
- Example:

```
xcode.configuration Main
```

xcode.target

If present, it overrides `build.target` and `destroot.target`.

- Type: optional
- Default: none
- Example:

```
xcode.target ${name}
```

xcode.build.settings

Additional settings passed to the xcode build tool during the build phase. These settings should be in the X=Y form.

- Type: optional
- Default: none
- Example:

```
xcode.build.settings FRAMEWORK_SEARCH_PATHS=${prefix}/Library/Frameworks
```

xcode.destroot.type

Type of project that will be installed. This tells the xcode PortGroup how to destroot the project. Correct values are `application` and `framework`.

- Type: optional
- Default: `application`
- Example:

```
xcode.destroot.type framework
```

xcode.destroot.path

Where to install the build product.

- Type: optional
- Default: `${frameworks_dir}` or `${applications_dir}` depending on `xcode.destroot.type`.

xcode.destroot.settings

Additional settings passed to the xcode build tool during the destroot phase. These settings should be in the X=Y form.

- Type: optional
- Default: none
- Example:

```
xcode.destroot.settings SKIP_INSTALL=NO
```

xcode.universal.settings

Settings passed to the xcode build tool when the +universal variant is selected. These settings should be in the X=Y form.

- Type: optional
- Default: `ARCHS="${universal_archs}" MACOSX_DEPLOYMENT_TARGET=${universal_target}`

xcode.universal.sdk

SDK to use when the +universal variant is selected. The argument may be an absolute path to an SDK, or the canonical name of an SDK.

- Type: optional
- Default: `${universal_sysroot}`

5.9.7.2. xcode PortGroup Sugar

Portfiles using the xcode PortGroup do not need to define the following variables:

categories

Default: aqua

platforms

Default: macosx

use_configure

Default: no

5.9.7.3. Portfile-Phase Keywords Affecting the xcode PortGroup

The following Portfile phase keywords affect the xcode PortGroup in a unique way. In most cases, you will not need to set any of these keywords in the Portfile. See portfile-phase(7)

build.cmd

Default: `${xcodebuildcmd}`.

build.target

Default: ""

This variable will be ignored if `xcode.target` is set.

build.args

Default: `build`

destroot.cmd

Default: `${xcodebuildcmd}`

destroot.target

Default: ""

This variable will be ignored if `xcode.target` is set.

Chapter 6. MacPorts Internals

This chapter contains information about the MacPorts file layout, configuration files, a few fundamental port installation concepts, and the MacPorts APIs.

6.1. File Hierarchy

Name

porthier — layout of the ports filesystems

Description

A map of the filesystem hierarchy used by MacPorts and the ports it installs. Much of it is based on hier(7).

${prefix}

The base of the MacPorts filesystem hierarchy.

Default: `/opt/local/`

bin/

Common utilities, programming tools, and applications.

etc/

System configuration files and scripts.

include/

Standard C include files.

lib/

Archive libraries.

libexec/

System daemons and system utilities (executed by other programs).

Library/Frameworks/

Native Mac OS X Library Frameworks

sbin/

System programs and administration utilities.

share/

Architecture-independent files.

doc/

Miscellaneous documentation.

examples/

Examples for users and programmers.

info/

GNU Info hypertext system.

locale/

Localization files.

man/

Manual pages.

misc/

Miscellaneous system-wide ASCII text files.

src/

Source code.

var/

Multi-purpose log, temporary, transient and spool files.

db/

Miscellaneous automatically generated system-specific database files.

macports/

MacPorts package building topdir.

build/

Where ports are built and destrooted.

distfiles/

Storage location for the distfiles of fetched ports.

packages/

Obsolete. Formerly contained archives (packages) of installed ports.

receipts/

Obsolete. Formerly contained the registry information and receipts for installed ports, in flat-file format.

registry/

Contains the registry database in sqlite format.

software/

The files for each installed port are stored here.

sources/

Holds the sources for the ports tree (the Portfiles) and also MacPorts base.

spool/

Directory containing output spool files.

log/

Miscellaneous system log files.

run/

System information files describing various information about the system since it was booted.

www/

Files to be served by an http server.

cgi-bin/

Directory for cgi executables.

/Applications/MacPorts/

Native Mac OS X applications.

SEE ALSO

port(1), macports.conf(5), portfile(7), portgroup(7), portstyle(7), hier(7)

AUTHORS

Felix Kroniage <fkr@opendarwin.org>

Juan Manuel Palacios <jmpp@macports.org>

6.2. Configuration Files

The MacPorts configuration files often do not need to be modified for the general end user. They contain options that may be of use to advanced users and port developers. Some automatically configured options may need to be updated when migrating to a new CPU architecture or a new OS version.

There are three MacPorts configuration files that define important variables used by the MacPorts system: `macports.conf`, `sources.conf`, and `variants.conf`. All MacPorts configurations files are located in `${prefix}/etc/macports`.

MacPorts configuration file format is a simple key/value pair separated by either a space or a tab. Lines beginning with '#' are comments, empty lines are ignored.

6.2.1. macports.conf

macports.conf is the configuration file used to bootstrap the MacPorts system. This file is read by the port command and determines how it behaves.

Options locating other .conf files.

sources_conf

Where to find the sources list.

Default: `${prefix}/etc/macports/sources.conf`

variants_conf

Where to find global variants definition file (optional).

Default: `${prefix}/etc/macports/variants.conf`

Options for MacPorts general operating characteristics.

prefix

Sets the directory where ports are installed. Any path may be used but those with spaces and/or non-ASCII characters should be avoided because it can break some ports.
Default: `/opt/local`

portdbpath

Directory where MacPorts keeps working data as downloaded sources, installed port receipts, and the main registry. Same path restrictions apply as for '${prefix}'.

Default: ${prefix}/var/macports

portdbformat

Formerly selected the storage type to use for the MacPorts registry: flat or sqlite. Currently, only sqlite can be used.

Default: `sqlite`

build_arch

The machine architecture to build for in normal use. Options include: ppc, i386, ppc64, x86_64

Default:

(Snow Leopard and later) `x86_64` or `i386` depending on hardware

(Leopard/Tiger) `i386` or `ppc` depending on hardware

applications_dir

Directory to install MacPorts that install Mac OS X .app bundles.

Default: `/Applications/MacPorts`

frameworks_dir

Directory to install frameworks installed by ports.

Default: `${prefix}/Library/Frameworks`

developer_dir

Directory where Xcode is installed.

Default: `/Developer`

portarchivetype

Type of archive to use for port images. Available types are: tgz, tar, tbz, tbz2, tlz, xar, zip, cpgz, cpio.

Default: `tbz2`

configureccache

Use ccache (C/C++ compiler cache) - see http://ccache.samba.org/

Default: `no`

configuredistcc

Use distcc (distributed compiler) - see http://distcc.samba.org/

Default: `no`

configurepipe

Use pipes rather than intermediate files when compiling C/C++/etc

Default: `yes`

buildnicevalue

Lowered scheduling priority (0-20) to use for make when building ports.

Default: 0

buildmakejobs

Number of simultaneous make jobs (commands) to use when building ports. Using "0" will cause a runtime autodetection to use all available processor cores.
Default: 0

portautoclean

Set whether to automatically execute "clean" after "install" of ports.

Default: `yes`

rsync_server

Rsync server from which to fetch MacPorts sources.

Default: rsync.macports.org

rsync_dir

Rsync directory from which to pull the base/ component (infrastructure) of MacPorts.

Default: `release/tarballs/base.tar`

rsync_options

Rsync options

Default: -rtzv --delete-after

destroot_umask

Umask value to use during the destrooting or a port.

Default: 022

binpath

Sets env(PATH), the directory search path for locating system executables (rsync, tar, etc.) during port installation. Only applications in these directories are available while ports are being installed even if other paths are specified by $PATH in a user's environment.

Default: ${prefix}/bin:${prefix}/sbin:/bin:/sbin:/usr/bin:/usr/sbin

> **Note**
>
> The binpath is implicitly defined, but it may be overwritten by defining the variable in macports.conf. However, using a non-default binpath is discouraged and should only be performed by advanced users.

Options for MacPorts Universal Binaries (+universal variant)

universal_archs

The machine architectures to use for +universal variant (multiple entries must be space delimited). Options include: ppc, i386, ppc64, x86_64

Default: `x86_64 i386` (`ppc i386` for 10.5 and earlier)

Options for StartupItems

startupitem_type

Options for generated startup items, though this may be overridden by the "startupitem.type" Portfile key. Options are "default" option, "SystemStarter", "launchd", or "none". For an empty or "default" option, a startupitem type appropriate to the platform is used; if "none", no port startupitems are installed.

Default: `default`

Other options

extra_env

Extra environment variables to keep. Any variables listed here are added to the list of variables that are not removed from the environment used while processing ports.

Default: none

place_worksymlink

Set whether to place a symlink named "work" from your ports tree to the build directory of a port, when the port

is being built. This is convenient, but may not be ideal if you care about the structure of your ports tree. For example, some developers keep their ports tree synchronized across multiple computers, and don't want to also synch build directories.

Default: yes

6.2.2. sources.conf

This file enables rsync synchronization of the default ports tree with the MacPorts rsync server when either of the commands **port selfupdate** or **port sync** are run.

Default: `rsync://rsync.macports.org/release/tarballs/ports.tar [default]`

Optional local repositories are enabled using a file url: *`file:///path/to/localportsrepository`*

6.2.3. variants.conf

This optional file specifies any variants you'd like to be invoked globally. If a variant specified ib this file is not supported by a given Portfile, the variant is simply ignored.

Default: none

6.3. Port Images

MacPorts has a unique ability to allow multiple versions, revisions, and variants of the same port to be installed at the same time, so you may test new port versions without uninstalling a previous working version.

This capability derives from the fact that a MacPorts port by default is not installed into its final or "activated" location, but rather to an intermediate location that is only made available to other ports and end-users after an activation phase that extracts all its files from the image repository. Deactivating a port only removes the files from their activated locations (usually under `${prefix}`) —the deactivated port's image is not disturbed.

The location of an installed port's image can be seen by running:

```
%% port location PORTNAME
```

6.4. APIs and Libs

The MacPorts system is composed of three Tcl libraries:

- MacPorts API - MacPorts public API for handling Portfiles, dependencies, and registry
- Ports API - API for Portfile parsing and execution
- pextlib - C extensions to Tcl

6.4.1. Ports API

The code for the Port API is located in `base/src/port1.0`. The Port API provides all the primitives required for a Portfile to be parsed, queried, and executed. It also provides a single procedure call that the MacPorts API uses to kick off execution: "eval_targets". The port Tcl library supplies these procedures, all of which are generated at run-time using the `options` procedure in portutil.tcl.

The macports Tcl library loads the Portfile into a sub-interpreter, within which all port-specific code is run. This process ensures that there will never be pollution of the Tcl space of other ports, nor the MacPorts libraries, nor the calling application.

> **Note**
>
> Portfiles are executed in a Tcl interpreter as Tcl code (and not truly parsed strictly speaking), so every Portfile option must be a TCL procedure.

The Ports API performs the following functions:

- Manages target registrations. All targets register themselves with the Port API. Accordingly, the Port API creates pre-/post-/main overrides for each of the targets.
- Option/Default handling. All Portfile options (name, version, revision ...) are registered by targets. The Port API creates procedures for these options, and sets up the complex variable traces necessary to support option defaults.
- Executes target procedures, including the pre/post/main routines.
- Manages a state file containing information about what variants were specified and what targets have run successfully.
- Provides essential Portfile Tcl extensions (reinplace, xinstall, etc).
- Provides simple access to the ui_event mechanism by providing the various ui_ procedures (i.e., ui_msg, ui_error).

6.4.2. MacPorts API

The code for the MacPorts API is located in `base/src/macports1.0`. The MacPorts API provides a public API into the MacPorts system by providing simple primitives for handling Portfiles, dependencies, and registry operations, and exports the MacPorts API for the **port** command line utility, or any other. The API has very little information about the contents Portfiles; instead, it relies entirely upon the **port** Tcl library. By keeping the high level API simple and generic, revisions to the underlying ports system will not necessarily require a revision of the high level MacPorts API.

The MacPorts API is also responsible for loading user specified options into a sub-interpreter to be evaluated by the ports API. In that case it sets the variable name in the sub-interpreter and adds the option to the sub-interpreter's global array user_options(). User options are passed as part of the call to `mportopen`.

The MacPorts API performs the following functions:

- Dependency support.

  This is implemented in a highly generic fashion, and is used throughout the system. The dependency functions are exported to the Port API, and the Port API uses them to execute targets in the correct order.

- Dependency processing.

  Software dependencies are handled at this layer using the dependency support layer.

- UI abstractions.

  UI Abstractions are handled at this layer. Each port action is provided a context, and a mechanism for posting user interface events is exported to the Port API (ui_event).

- Registry management routines.

  Manages the rudimentary port registry in `${prefix}/var/macports/receipts/`.

  - `mportregistry::new`: create a new port registry entry.
  - `mportregistry::exists`: check if a port registry entry exists (either versioned or not).
  - `mportregistry::delete`: delete an existing registry entry.
  - `mportregistry::close`: closes a new registry entry.

- Exports the MacPorts API for use by client applications.

  The following routines are defined.

  - `mportinit`: Initializes the MacPorts system. Should be called before trying to use any other procedure.
  - `mportsearch`: Given a regexp, searches the `PortIndex` for ports with matching names.
  - `mportopen`: Given a URI to a port, opens a Portfile and returns an opaque handle to it.
  - `portclose`: Given a port handle, closes a Portfile.
  - `mportexec`: Given a port handle, executes a target (i.e. install).
  - `mportinfo`: Given a port handle, this returns the PortInfo array (as a flat list of array elements). This is a little tricky and unstable and only used by the **portindex** command.
  - `mportdepends`: Given a port handle, returns a list of ports upon which the specified port depends.

For an example of the MacPorts API, when one executes **port search cm3**, the port utility:

- Calls the `mportsearch` function to find all ports containing "cm3".
- Returns Tcl array(s) containing data from the `PortIndex`: port name, version, revision, variants, etc.
- Formats the list of arrays in the standard viewing format.

For another MacPorts API example, when one executes **port install cm3**, the port utility:

- Calls the `mportsearch` function to find the first port that matches the name "cm3".
- Calls the `mportopen` function to open the port.
- Calls the `mportexec` function to execute the install target in the port.

- Calls the `mportclose` function to close the port.

6.4.3. pextlib

The pextlib TCL library provides a variety of C extensions to add capabilities to TCL procedures; for example, an interface to flock(2) and mkstemp(3).

6.5. The MacPorts Registry

This chapter provides an overview of the MacPorts registry and its API. The registry is queried by MacPorts utilities for information about installed ports related to dependencies, port images, and simple user information about what is installed. It provides abstraction over a modular receipt storage layer; this allows for flat file receipts as well as receipts stored in a SQLite database.

The registry allows MacPorts utilities to:

- Modify receipts to reflect changes made to installed ports being maintained by MacPorts.
- Query the global file and dependency databases for file conflicts between a port being installed and a port already installed.
- Maintain dependency trees of installed ports.

6.5.1. Registry Files

The flat file registry (MacPorts default registry) files are contained in `${portdbpath}/receipts`, which by default is location `${prefix}/var/macports/receipts`. File mappings and dependency mappings are tracked in the flat file registry by two files:

- file_map.db
- dep_map.bz2

6.5.2. The Registry API

The MacPorts registry provides a public API in the registry1.0 Tcl package. Using this API listed below you can access the MacPorts Registry using the default receipt storage mechanism chosen in `macports.conf`.

`registry::new_entry {name version {revision 0} {variants ""}}`

Begin the creation of a new registry entry for the given port. Returns a reference ID to the registry entry created.

`registry::open_entry {name {version 0} {revision 0} {variants ""}}`

Opens an existing registry entry. Returns a reference ID to the registry entry that was opened.

`registry::entry_exists {name version {revision 0} {variants ""}}`

Checks to see if a port exists in the registry. Returns 1 if the entry exists, 0 if not.

`registry::write_entry {ref}`

Writes the receipt associated with the given reference.

`registry::delete_entry {ref}`

Deletes the receipt associated with the given reference.

`registry::property_store {ref property value}`

Store the given value with the property name in the receipt associated with the given reference.

`registry::property_retrieve {ref property}`

Retrieve the property name from the receipt associated with the given reference. Returns the value of the property, if the property exists.

`registry::installed {{name ""} {version ""}}`

Get all installed ports, optionally all installed ports matching the given name, or the given name and version. Returns a list of the installed ports.

`registry::location {portname portversion}`

Returns the physical location the port is installed in on the disk. This is primarily useful for finding out where a

port image is installed.

registry::open_file_map {args}

Opens the file map that contains file-port relationships.

registry::file_registered {file}

Returns the name of the port that owns the given file, if the file is registered as installed, and 0 otherwise.

registry::port_registered {name}

Returns a list of all files associated with the given port if that port is installed, and 0 otherwise.

registry::register_file {file port}

Registers the given file in the file map as belonging to the given port.

registry::unregister_file {file}

Removes the file from the file map.

registry::write_file_map {args}

Write the changes to the file map.

registry::open_dep_map {args}

Opens the dependency map that contains port dependency relationships.

registry::fileinfo_for_file {fname}

Returns a list for the given file name representing all data currently known about the file. This is a 6-tuple in the form of:

1. file path
2. uid
3. gid
4. mode
5. size
6. md5 checksum

registry::fileinfo_for_index {flist}

Returns a list of information concerning each file in the given file list, if that file exists in the registry. The information if obtained through registry::fileinfo_for_file

registry::list_depends {name}

Returns a list of all the ports that given port name depends on.

registry::list_dependents {name}

Returns a list of all the ports that depend on the given port name.

registry::register_dep {dep type port}

Registers the given dependency as the given type of dependency with the given port.

registry::unregister_dep {dep type port}

Unregister the given dependency of the given type as a dependency of the given port.

registry::write_dep_map {args}

Write changes to the dependency map.

Chapter 7. MacPorts Project

7.1. Using Trac for tickets

The MacPorts Project uses a system called Trac to file tickets to report bugs and enhancement requests. Trac also provides an interface to browse the MacPorts Subversion repository. Though anyone may search Trac for tickets, you must register for a Trac account to create tickets.

must register for a Trac account to create tickets.

7.1.1. Before you file a new ticket

- Check the Problem HotList

  The Problem Hotlist contains possible solutions to problems that affect many MacPorts users. If a solution to your problem listed there works, don't file a ticket.

- Search to see if a Trac ticket has already been filed

  Avoid filing duplicate bugs. Search for duplicates by:

  - using the search bar that appears on each page
  - using the search page
  - browsing the list of categorised reports
  - making an advanced search by constructing a custom query

- Is the problem an application error and not related to compiling and installing?

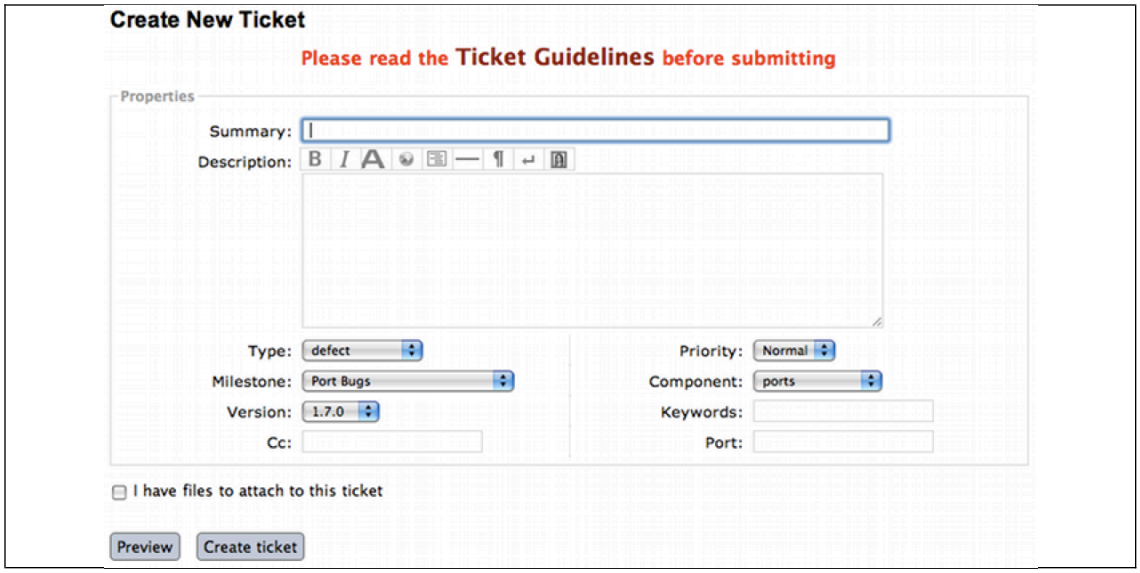
  In general, application bugs should be reported to the developers of the app ("upstream"), not MacPorts. An application bug that affects a large number of MacPorts users might merit a MacPorts bug for informational purposes only, but this should be done sparingly.

- Is the problem with a 'port upgrade' operation?

  If so, try a 'port uninstall *foo*' and then reinstall. You might also want to run 'port -nR upgrade --force *foo*' to rebuild ports depending upon port *foo*.

7.1.2. Creating Trac Tickets

Once you are logged into Trac, you may click New Ticket and you will be presented with a new ticket window shown in the graphic below. Follow the Trac ticket guidelines below to fill out the form. If you are reporting a failed port install and a log was mentioned in the error, please use the "I have files to attach to this ticket" checkbox to add that log file to the ticket.



7.1.3. Trac Ticket Guidelines

There are certain conventions used to ensure that Trac tickets convey as much accurate information as possible so problems and contributions may be acted upon efficiently.

- **Summary:** *[port] [version] [concise description]*
  - Example: "rrdtool @1.2.23 +python Configure error - build failure"
- **Description:** All details that might be relevant to someone reading the ticket. Be sure to mention the versions of your operating system and Xcode install. Wiki formatting should be used to ensure that text is formatted correctly. Use the Preview button before submitting. If you want to post preformatted text such as a log or terminal output, make sure you use {{{...}}} around the text or it could break the page layout. Example:

terminal output, make sure you use {{{ }}} around the text or it could break the page layout. Example:

```
{{{
your error message here
}}}
```

Submitters are advised to trim inline pastes and logs to what's really relevant to the report, as otherwise overly large ticket pages can become unmanageable. Long output, such as the full log from a port build, should be added as an attachment, not pasted inline. See "I have files to attach to this ticket" below.

- **Type:** There are five types of tickets.
  - **defect** - The default; any port/MacPorts build/runtime failures and/or documentation corrections.
  - **enhancement** - Tickets, with or without patches, created to enhance something that isn't failing its intended purpose.
  - **update** - Tickets, with or without patches, involving updating a port to a newer upstream version.
  - **submission** - Tickets created to submit Portfiles for software not currently available in MacPorts.
  - **request** - Tickets created to request the creation of a new port.
- **Priority:** Assign a priority level to the ticket.
  - **High** - Reserved for the use of MacPorts team members, as they are the best fit to determine which reports warrant a higher priority over others.
  - **Normal** - The default. For normal port failures, non-critical enhancement requests, non-critical port failures.
  - **Low** - For mostly cosmetic improvements, documentation corrections/improvements, etc.
  - **Not set** - Anything that doesn't fit the categories high, normal, or low.
- **Milestone:** This is a ticket label that indicates that the ticket is intended to be fixed in a particular MacPorts release. Leave it blank; it will be set by a project member if appropriate.
- **Component:** Set what part of the MacPorts Project the ticket is to be filed against.
  - **base** - Tickets related to MacPorts base code.
  - **guide** - Documentation enhancements and error corrections, or patches to the MacPorts Guide.
  - **ports** - Tickets related to ports.
  - **server/hosting** - For MacPorts hosting & server-side issues, reserved for MacPorts PortMgr team members.
  - **website** - MacPorts website enhancements and error corrections.
  - **wiki** - MacPorts Wiki enhancements and error corrections.
- **Version:** Select the MacPorts version you are using when it is applicable.
- **Keywords:** Type any keywords that might help when searching for tickets. It is not useful to list words here that already appear elsewhere in the ticket. Keywords also serve as tags; for example, use "tiger" if reporting a bug that only affects OS X 10.4, "haspatch" if a fix is attached to the ticket, "maintainer" if you are the port's maintainer, or "LP64" if reporting an issue that only affects 64-bit platforms.
- **Cc:** Anyone else besides the ticket reporter and assignee who would like to be kept involved in the development of the ticket. Multiple email addresses should be separated with a comma and a space (i.e. `you@example.org`, `maintainer@macports.org`).

  Most users will not be able to assign tickets. If this applies to you, then when reporting port-related tickets, make sure you add the port's maintainer to cc. Otherwise they may not notice the ticket.
- **Assign To:** Only users with commit access can edit this field. If this is not you, see the section on the **Cc** field above.

  For tickets on ports, enter the email address of the port's maintainer (use **port info <portname>** to find this). If multiple maintainers are listed, enter the first maintainer's email address here and enter the remainining maintainers' email addresses in the **Cc** field. Exclude the email address <openmaintainer@macports.org> if it appears. If the maintainer's email address is <nomaintainer@macports.org>, leave the field blank.
- **Port:** For tickets on ports, enter the name of the port (or ports, space-separated, when multiple are affected).

- **I have files to attach to this ticket:** Use this checkbox to attach files to the ticket immediately after you create it. Or you can attach files later using the **Attach File** button.

  If the file you are attaching is larger than 256 KiB, please compress it with bzip2 or gzip first to save space on the server and bandwidth for those downloading it, as Trac will not preview files above that size anyway.

7.2. Contributing to MacPorts

You may contribute new ports and enhancements of any kind to already existing ports using Trac tickets.

7.2.1. New Ports

Ports are contributed by following these steps. See Ticket Submission Guidelines above for a description of all fields.

1. Create a Trac ticket.
2. Set the type to **submission**.
3. Set the component to **ports**.
4. Attach the `Portfile` and any required patchfiles to the ticket.

7.2.2. Port Enhancements

Enhancements to existing ports may comprise new functionality for a given port, bug fixes or even simple version updates. They should always be contributed as `Portfile` patches. See Ticket Submission Guidelines above for a description of all fields.

1. Create a `Portfile` patch with your changes as described in Portfile Development.
2. Create a Trac ticket.
3. Set the type to **enhancement** for miscellaneous enhancements, to **defect** for bug fixes, or to **update** for version updates.
4. Set the component to **ports**.
5. Attach your Portfile patch file and any new or changed patch files (don't patch patches) to the ticket.

7.3. Port Update Policies

Port maintainers normally are given commit privileges to the Subversion repository so they can make updates to their own ports. However, The MacPorts Project does not restrict commit privileges for maintainers, so before a person other than a port's maintainer updates a port it is a good practice to inform a port's maintainer. See details below.

7.3.1. Non-Maintainer Port Updates

If you have a port update or bugfix for a port you do not maintain, to respect the rights of the port maintainer you should follow the following guidelines:

1. If a port's maintainer is <nomaintainer@macports.org>, you may feel free to make updates and/or take maintainership of the port.
2. If a port's maintainer contains the address <openmaintainer@macports.org>, this means that the author allows minor updates to the port without contacting him first. But permission should still be sought for major changes.
3. Create patch file(s) as necessary, attach them to a Trac ticket, and assign the ticket to the maintainer (or Cc him or her, if you are unable to assign tickets).
4. Wait for a response from the maintainer. The maintainer should apply the patches and close the ticket within 72 hours.

However, for maintained ports without <openmaintainer@macports.org>, there are some conditions under which maintainer permission may be waived:

- If the maintainer does not respond within 72 hours, you or another committer may review the patches and update the port. The log message of this commit must explain that you are taking advantage of maintainer timeout and include a reference to the ticket. If you are not a committer, you may email <macports-dev@lists.macosforge.org> and request the updates be committed.
- A port is abandoned by its current maintainer. A port against which a Port Abandoned ticket has been filed (see below) can be updated without contacting the maintainer.

- A critical port is broken that affects many users.

7.3.2. Port Abandonment

A port may be considered abandoned if a bug has not been acknowledged for more than three weeks after a ticket is filed. If this time period has passed and you wish to initiate the Port Abandonment protocol and volunteer as the new maintainer:

1. File a new Trac ticket with the summary line [Port Abandoned] **portname**.
2. Refer to the original unacknowledged ticket in the Port Abandoned ticket.
3. Further indicate which port is abandoned via the port field in the Port Abandoned ticket.
4. The Port Abandoned ticket may be closed when the new maintainer is assigned, and the original ticket with the updates may be resolved when the updates attached to the original ticket are committed.

7.4. MacPorts Membership

A requirement for a person to become a MacPorts committer is to first become involved and contribute to the project. This may be done by having a record of contribution to the project in several of the following ways:

- Contributing new ports.
- Fixing bugs in existing ports.
- Volunteering as a maintainer of non-maintained ports.
- Involvement on MacPorts support lists.
- Contributing with documentation

To apply for MacPorts commit rights, send a brief email to the PortMgr team at <macports-mgr@lists.macosforge.org> entitled "Commit access: *Your Name*" with the following contents:

- a description of your application and why you think you deserve commit rights (including evidence of contributions to MacPorts as described above).
- the identity you'd like to use as a member of the project, A.K.A. the "handle", as part of your *handle*@macports.org alias.
- a real e-mail address to which you'd like your MacPorts alias to forward.

The PortMgr team will consider all applications and provide an appropriate response in a timely manner.

7.5. The PortMgr Team

The MacPorts PortMgr team is the steering group for The MacPorts Project. Its membership is usually determined by public elections among project members; the current members of the team can be found on the MacPorts Developers wiki page. They are responsible for matters such as:

- approving new project members (i.e. granting commit rights);
- setting general guidelines for the project;
- dispute resolution;
- managing the projects infrastructure; and
- engineering releases.

Chapter 8. MacPorts Guide Terms

Glossary

MacPorts Guide Terms

activate phase

automake

autoconf

API

BSD Unix

CVS

destroot phase

port binary

build

build phase

checksum

checksum phase

compile

configure

configure phase

dependency

destroot phase

diff

extract phase

fetch phase

free software

global keyword

gunzip

keyword

keyword argument modifier

keyword list modifier

library

MacPorts

> A system for compiling, installing, and managing free and open source software comprised of an infrastructure called MacPorts base and a collection of ports. MacPorts current port collection defines the software may be installed.

open source software

patch phase

patch file

pextlib

phase

port

port command

port image

port maintainer

port phase

port phase keyword

PortGroup

Portfile

registry

rsync

selfupdate

shell

StartupItem

Subversion

sync

tar

Tcl

Tcl extension

Trac

Unix

unzip

variant

Xcode Tools

X11

zip