FreeBSD Porter's Handbook

The FreeBSD Documentation Project

FreeBSD Porter's Handbook

by

Revision: 43218
2013-11-22 by rene.

Copyright © 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013 The FreeBSD Documentation Project

Copyright

Redistribution and use in source (XML DocBook) and 'compiled' forms (XML, HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code (XML DocBook) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

2. Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Important

THIS DOCUMENTATION IS PROVIDED BY THE FREEBSD DOCUMENTATION PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FREEBSD DOCUMENTATION PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

FreeBSD is a registered trademark of the FreeBSD Foundation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Sun, Sun Microsystems, Java, Java Virtual Machine, JDK, JRE, JSP, JVM, Netra, OpenJDK, Solaris, StarOffice, SunOS and VirtualBox are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and the FreeBSD Project was aware of the trademark claim, the designations have been followed by the “™” or the “®” symbol.

Table of Contents

1. Introduction 1
2. Making a New Port Yourself 3
3. Quick Porting 5
    3.1. Writing the `Makefile` 5
    3.2. Writing the Description Files 6
    3.3. Creating the Checksum File 9
    3.4. Testing the Port 9
    3.5. Checking Your Port with `portlint` 10
    3.6. Submitting the New Port 10
4. Slow Porting 13
    4.1. How Things Work 13
    4.2. Getting the Original Sources 14
    4.3. Modifying the Port 15
    4.4. Patching 16
    4.5. Configuring 17
    4.6. Handling User Input 17
5. Configuring the Makefile 19
    5.1. The Original Source 19
    5.2. Naming 19
    5.3. Categorization 26
    5.4. The Distribution Files 33
    5.5. `MAINTAINER` 46
    5.6. `COMMENT` 47
    5.7. `PORTSCOUT` 47
    5.8. Dependencies 48
    5.9. `MASTERDIR` 55
    5.10. Man Pages 56
    5.11. Info Files 56
    5.12. Makefile Options 57
    5.13. Specifying the Working Directory 67
    5.14. Conflict Handling 68
    5.15. Installing Files 69
6. Special Considerations 73
    6.1. Staging 73
    6.2. Shared Libraries 74
    6.3. Ports with Distribution Restrictions 74
    6.4. Building Mechanisms 77
    6.5. Using GNU Autotools 80
    6.6. Using GNU `gettext` 82
    6.7. Using Perl 83
    6.8. Using X11 85
    6.9. Using GNOME 88
    6.10. Using Qt 88
    6.11. Using KDE 92

6.12. Using Java 93
6.13. Web Applications, Apache and PHP 97
6.14. Using Python 100
6.15. Using Tcl/Tk 102
6.16. Using Emacs 103
6.17. Using Ruby 103
6.18. Using SDL 104
6.19. Using wxWidgets 105
6.20. Using Lua 111
6.21. Using `iconv` 117
6.22. Using Xfce 119
6.23. Using Mozilla 120
6.24. Using Databases 121
6.25. Starting and Stopping Services (rc Scripts) 122
6.26. Adding Users and Groups 125
6.27. Ports That Rely on Kernel Sources 125
7. Advanced `pkg-plist` Practices 127
7.1. Changing `pkg-plist` Based on Make Variables 127
7.2. Empty Directories 128
7.3. Configuration Files 129
7.4. Dynamic Versus Static Package List 129
7.5. Automated Package List Creation 130
8. The `pkg-*` Files 131
8.1. `pkg-message` 131
8.2. `pkg-install` 131
8.3. `pkg-deinstall` 132
8.4. Changing the Names of `pkg-*` Files 132
8.5. Making Use of `SUB_FILES` and `SUB_LIST` 133
9. Testing Your Port 135
9.1. Running `make describe` 135
9.2. Portlint 135
9.3. Port Tools 135
9.4. `PREFIX` and `DESTDIR` 136
9.5. Tinderbox 137
10. Upgrading an Individual Port 139
10.1. Using `SVN` to Make Patches 141
10.2. The Files `UPDATING` and `MOVED` 142
11. Ports Security 143
11.1. Why Security is So Important 143
11.2. Fixing Security Vulnerabilities 143
11.3. Keeping the Community Informed 144
12. Dos and Don'ts 151
12.1. Introduction 151
12.2. `WRKDIR` 151
12.3. `WRKDIRPREFIX` 151
12.4. Differentiating Operating Systems and OS Versions 151

12.5. Writing Something After bsd.port.mk 152
12.6. Use the exec Statement in Wrapper Scripts 153
12.7. Do Things Rationally 153
12.8. Respect Both CC and CXX 154
12.9. Respect CFLAGS 154
12.10. Threading Libraries 155
12.11. Feedback 155
12.12. README.html 155
12.13. Marking a Port Not Installable with BROKEN, FORBIDDEN, or IGNORE 156
12.14. Marking a Port for Removal with DEPRECATED or EXPIRATION_DATE 158
12.15. Avoid Use of the .error Construct 158
12.16. Usage of sysctl 159
12.17. Rerolling Distfiles 159
12.18. Avoiding Linuxisms 159
12.19. Miscellanea 160
13. A Sample Makefile 161
14. Keeping Up 163
14.1. FreshPorts 163
14.2. The Web Interface to the Source Repository 163
14.3. The FreeBSD Ports Mailing List 163
14.4. The FreeBSD Port Building Cluster 164
14.5. Portscout: the FreeBSD Ports Distfile Scanner 164
14.6. The FreeBSD Ports Monitoring System 164
15. Appendices 167
15.1. Values of USES 167
15.2. __FreeBSD_version Values 176

List of Tables

5.1. Popular Magic MASTER_SITES Macros 35
5.2. The USE_* Variables 52
5.3. Common WITH_* and WITHOUT_* Variables 58
6.1. Variables for Ports Related to gmake 77
6.2. Variables for Ports That Use configure 78
6.3. Variables for Ports That Use cmake 78
6.4. Variables for Ports That Use scons 79
6.5. Variables for Ports That Use Perl 84
6.6. Variables for Ports That Use X 86
6.7. Variables for Ports That Use Qt 88
6.8. Additional Variables for Ports That Use Qt 4.x 89
6.9. Available Qt 4 Library Components 89
6.10. Available Qt 4 Tool Components 90
6.11. Available Qt 4 Plugin Components 90
6.12. Variables for Ports That Use qmake 91
6.13. Available KDE 4 Components 92
6.14. Variables Which May be Set by Ports That Use Java 94
6.15. Variables Provided to Ports That Use Java 94
6.16. Constants Defined for Ports That Use Java 96
6.17. Variables for Ports That Use Apache 97
6.18. Useful Variables for Porting Apache Modules 98
6.19. Variables for Ports That Use PHP 99
6.20. Most Useful Variables for Ports That Use Python 100
6.21. The Most Useful Variables for Ports That Use Tcl/Tk 102
6.22. Useful Variables for Ports That Use Ruby 103
6.23. Selected Read-Only Variables for Ports That Use Ruby 103
6.24. Variables to Select wxWidgets Versions 106
6.25. Available wxWidgets Versions 106
6.26. wxWidgets Version Specifications 106
6.27. Variables to Select Preferred wxWidgets Versions 107
6.28. Available wxWidgets Components 107
6.29. Available wxWidgets Dependency Types 107
6.30. Default wxWidgets Dependency Types 108
6.31. Variables to Select Unicode in wxWidgets Versions 108
6.32. Variables Defined for Ports That Use wxWidgets 110
6.33. Legal Values for WX_CONF_ARGS 111
6.34. Variables to Select Lua Versions 112
6.35. Available Lua Versions 112
6.36. Lua Version Specifications 112
6.37. Variables to Select Preferred Lua Versions 112
6.38. Available Lua Components 113
6.39. Available Lua Dependency Types 114
6.40. Default Lua Dependency Types 114
6.41. Variables Defined for Ports That Use Lua 115

6.42. Variables for Ports That Use Mozilla 120
6.43. Variables for Ports Using Databases 121
10.1. SVN Update File Prefixes 141
15.1. Values of USES 167
15.2. __FreeBSD_version Values 176

List of Examples

5.1. Simplified Use of MASTER_SITES:n with One File Per Site 39
5.2. Simplified Use of MASTER_SITES:n with More Than One File Per Site 39
5.3. Detailed Use of MASTER_SITES:n in MASTER_SITE_SUBDIR 41
5.4. Detailed Use of MASTER_SITES:n with Comma Operator, Multiple Files, Multiple Sites and Multiple Subdirectories 41
5.5. Detailed Use of MASTER_SITES:n with MASTER_SITE_SOURCEFORGE 43
5.6. Simplified Use of MASTER_SITES:n with PATCH_SITES 44
5.7. Use of ALWAYS_KEEP_DISTFILES 46
5.8. Wrong Declaration of an Optional Dependency 54
5.9. Correct Declaration of an Optional Dependency 54
5.10. Simple Use of OPTIONS 60
5.11. Check for Unset Port OPTIONS 60
5.12. Practical Use of OPTIONS 60
5.13. Wrong Handling of an Option 62
5.14. Correct Handling of an Option 62
6.1. USES= cmake Example 79
6.2. Perl Dependency Example 85
6.3. USE_XORG Example 86
6.4. Using X11-Related Variables 86
6.5. Selecting Qt 4 Components 90
6.6. USE_KDE4 Example 93
6.7. Example Makefile for PEAR Class 99
6.8. Selecting wxWidgets Components 108
6.9. Detecting Installed wxWidgets Versions and Components 109
6.10. Using wxWidgets Variables in Commands 110
6.11. Selecting the Lua Version 113
6.12. Selecting Lua Components 114
6.13. Detecting Installed Lua Versions and Components 114
6.14. Telling the Port Where to Find Lua 116
6.15. Using Lua Variables in Commands 117
6.16. Simple iconv Usage 118
6.17. iconv Usage with configure 118
6.18. Fixing Hardcoded -liconv 119
6.19. Checking for Native iconv Availability 119
12.1. How to Avoid Using .error 158

Chapter 1. Introduction

The FreeBSD ports collection is the way almost everyone installs applications ("ports") on FreeBSD. Like everything else about FreeBSD, it is primarily a volunteer effort. It is important to keep this in mind when reading this document.

In FreeBSD, anyone may submit a new port, or volunteer to maintain an existing port if it is unmaintained—you do not need any special commit privileges to do so.

Chapter 2. Making a New Port Yourself

So, you are interested in making your own port or upgrading an existing one? Great!

What follows are some guidelines for creating a new port for FreeBSD. If you want to upgrade an existing port, you should read this and then read Chapter 10, *Upgrading an Individual Port*.

When this document is not sufficiently detailed, you should refer to `/usr/ports/Mk/bsd.port.mk` , which all port Makefiles include. Even if you do not hack Makefiles daily, it is well commented, and you will still gain much knowledge from it. Additionally, you may send specific questions to the FreeBSD ports mailing list.

Note

Only a fraction of the variables (*VAR*) that can be overridden are mentioned in this document. Most (if not all) are documented at the start of `/usr/ports/Mk/bsd.port.mk` ; the others probably ought to be. Note that this file uses a non-standard tab setting: Emacs and Vim should recognize the setting on loading the file. Both vi(1) and ex(1) can be set to use the correct value by typing `:set tabstop=4` once the file has been loaded.

Looking for something easy to start with? Take a look at the list of requested ports and see if you can work on one (or more).

Chapter 3. Quick Porting

This section tells you how to quickly create a new port. In many cases, it is not sufficient, so you will have to read further on into the document.

First, get the original tarball and put it into `DISTDIR`, which defaults to `/usr/ports/distfiles`.

Note

The following assumes that the software compiled out-of-the-box, i.e., there was absolutely no change required for the port to work on your FreeBSD box. If you needed to change something, you will have to refer to the next section too.

Note

It is recommended to set the `DEVELOPER` make(1) variable in `/etc/make.conf` before getting into porting.

```
# echo DEVELOPER=yes >> -/etc/make.conf
```

This setting enables the “developer mode” that displays deprecation warnings and activates some further quality checks on calling the `make` command.

3.1. Writing the `Makefile`

The minimal `Makefile` would look something like this:

```
# $FreeBSD$

PORTNAME= oneko
PORTVERSION= 1.1b
CATEGORIES= games
MASTER_SITES= ftp://ftp.cs.columbia.edu/archives/X11R5/contrib/

MAINTAINER= youremail@example.com
COMMENT= Cat chasing a mouse all over the screen
.include <bsd.port.mk>
```

Note

In some cases, the `Makefile` of an existing port may contain additional lines in the header, such as the name of the port and the date it was created. This additional information has been declared obsolete, and is being phased out.

See if you can figure it out. Do not worry about the contents of the `$FreeBSD$` line, it will be filled in automatically by SVN when the port is imported to our main ports tree. You can find a more detailed example in the sample Makefile section.

3.2. Writing the Description Files

There are two description files that are required for any port, whether they actually package or not. They are `pkg-descr` and `pkg-plist`. Their `pkg-` prefix distinguishes them from other files.

3.2.1. `pkg-descr`

This is a longer description of the port. One to a few paragraphs concisely explaining what the port does is sufficient.

Note

This is *not* a manual or an in-depth description on how to use or compile the port! *Please be careful if you are copying from the `README` or manpage*; too often they are not a concise description of the port or are in an awkward format (e.g., manpages have justified spacing, as it looks particularly bad with monospaced fonts).

A well-written `pkg-descr` describes the port completely enough that users would not have to consult the documentation or visit the website to understand what the software does, how it can be useful, or what particularly nice features it has. Mentioning certain requirements like a graphical toolkit, heavy dependencies, runtime environment, or implementation languages help users decide whether this port will work for them.

Include a URL to the official WWW homepage. Prepend *one* of the websites (pick the most common one) with `WWW:` (followed by single space) so that automated tools will work cor-

rectly. If the URI is the root of the website or directory, it should be terminated with a slash.

Note

If the listed webpage for a port is not available, try to search the Internet first to see if the official site moved, was renamed, or is hosted elsewhere.

The following example shows how your `pkg-descr` should look:

```
This is a port of oneko, in which a cat chases a poor mouse all over
the screen.
 -:
(etc.)

WWW: http://www.oneko.org/
```

3.2.2. `pkg-plist`

This file lists all the files installed by the port. It is also called the "packing list" because the package is generated by packing the files listed here. The pathnames are relative to the installation prefix (usually `/usr/local` . If the port creates directories during installation, make sure to add `@dirrm` lines to remove them when the package is deleted.

Here is a small example:

```
bin/oneko
man/man1/oneko.1.gz
lib/X11/app-defaults/Oneko
lib/X11/oneko/cat1.xpm
lib/X11/oneko/cat2.xpm
lib/X11/oneko/mouse.xpm
@dirrm lib/X11/oneko
```

Refer to the pkg_create(1) manual page for details on the packing list.

Note

It is recommended that you keep all the filenames in this file sorted alphabetically. It will make verifying the changes when you upgrade the port much easier.

Note

Creating a packing list manually can be a very tedious task. If the port installs a large numbers of files, creating the packing list automatically might save time.

There is only one case when `pkg-plist` can be omitted from a port. If the port installs just a handful of files, and perhaps directories, the files and directories may be listed in the variables `PLIST_FILES` and `PLIST_DIRS`, respectively, within the port's `Makefile`. For instance, we could get along without `pkg-plist` in the above `oneko` port by adding the following lines to the `Makefile`:

```
PLIST_FILES= bin/oneko \
  man/man1/oneko.1.gz \
  lib/X11/app-defaults/Oneko \
  lib/X11/oneko/cat1.xpm \
  lib/X11/oneko/cat2.xpm \
  lib/X11/oneko/mouse.xpm
PLIST_DIRS= lib/X11/oneko
```

Of course, `PLIST_DIRS` should be left unset if a port installs no directories of its own.

Note

Several ports can share a common directory. In that case, `PLIST_DIRS` should be replaced by `PLIST_DIRSTRY` so that the directory is removed only if empty, otherwise it is silently ignored. `PLIST_DIRS` and `PLIST_DIRSTRY` are equivalent to using `@dirrm` and `@dirrmtry` in `pkg-plist`, as described in Section 7.2.1, "Cleaning Up Empty Directories".

The price for this way of listing port's files and directories is that you cannot use command sequences described in pkg_create(1). Therefore, it is suitable only for simple ports and makes them even simpler. At the same time, it has the advantage of reducing the number of files in the ports collection. Please consider using this technique before you resort to `pkg-plist`.

Later we will see how `pkg-plist` and `PLIST_FILES` can be used to fulfill more sophisticated tasks.

3.3. Creating the Checksum File

Just type `make makesum`. The ports make rules will automatically generate the file `distinfo`.

If a file fetched has its checksum changed regularly and you are certain the source is trusted (i.e., it comes from manufacturer CDs or documentation generated daily), you should specify these files in the `IGNOREFILES` variable. Then the checksum is not calculated for that file when you run `make makesum`, but set to `IGNORE`.

3.4. Testing the Port

You should make sure that the port rules do exactly what you want them to do, including packaging up the port. These are the important points you need to verify.

- `pkg-plist` does not contain anything not installed by the port.
- `pkg-plist` contains everything that is installed by the port.
- The port can be installed using the `install` target. This verifies that the install script works correctly.
- The port can be deinstalled properly using the `deinstall` target. This verifies that the deinstall script works correctly.
- Make sure that `make package` can be run as a normal user (that is, not as `root`). If that fails, `NEED_ROOT=yes` must be added to the port `Makefile`.

Procedure 3.1. Recommended Test Ordering

1. `make stage`
2. `make check-orphans`
3. `make package`
4. `make install`
5. `make deinstall`
6. `pkg_add package-name`

   Or, for users of *pkg*:

   `pkg add package-name`

7. `make package` (as user)

Make certain no warnings are shown in any of the stages.

Thorough automated testing can be done with ports-mgmt/tinderbox or ports-mgmt/poudriere from the Ports Collection. These applications maintain `jails` where all of the steps shown above can be tested without affecting the state of the host system.

3.5. Checking Your Port with `portlint`

Please use `portlint` to see if your port conforms to our guidelines. The ports-mgmt/portlint program is part of the ports collection. In particular, you may want to check if the Makefile is in the right shape and the package is named appropriately.

3.6. Submitting the New Port

Before you submit the new port, make sure you have read the DOs and DON'Ts section.

Now that you are happy with your port, the only thing remaining is to put it in the main FreeBSD ports tree and make everybody else happy about it too. We do not need your `work` directory or the `pkgname.tgz` package, so delete them now. Next, assuming your port is called oneko, `cd` to the directory above where the `oneko` directory is located, and then type the following: `shar `find oneko` > oneko.shar`

Include your `oneko.shar` file in a bug report and send it with the send-pr(1) program (see Bug Reports and General Commentary for more information about send-pr(1)). Be sure to classify the bug report as category `ports` and class `change-request` (Do not mark the report `confidential`!). Also add a short description of the program you ported to the "Description" field of the PR (e.g., perhaps a short version of the `COMMENT`), and add the shar file to the "Fix" field.

Note

You can make our work a lot easier, if you use a good description in the synopsis of the problem report. We prefer something like "New port: <category>/<portname> <short description of the port>" for new ports. If you stick to this scheme, the chance that someone will take a look at your PR soon is much better.

One more time, *do not include the original source distfile, the `work` directory, or the package you built with `make package`*; and, do use shar(1) for new ports, not diff(1).

After you have submitted your port, please be patient. Sometimes it can take a few months before a port is included in FreeBSD, although it might only take a few days. You can view the list of ports PRs waiting to be committed to FreeBSD.

Once we have looked at your port, we will get back to you if necessary, and put it in the tree. Your name will also be added to the list of Additional FreeBSD Contributors and other files.

Chapter 4. Slow Porting

Ok, so it was not that simple, and the port required some modifications to get it to work. In this section, we will explain, step by step, how to modify it to get it to work with the ports paradigm.

4.1. How Things Work

First, this is the sequence of events which occurs when the user first types `make` in your port's directory. You may find that having `bsd.port.mk` in another window while you read this really helps to understand it.

But do not worry if you do not really understand what `bsd.port.mk` is doing, not many people do... *:-)*

1. The `fetch` target is run. The `fetch` target is responsible for making sure that the tarball exists locally in `DISTDIR`. If `fetch` cannot find the required files in `DISTDIR` it will look up the URL `MASTER_SITES`, which is set in the Makefile, as well as our FTP mirrors where we put distfiles as backup. It will then attempt to fetch the named distribution file with `FETCH`, assuming that the requesting site has direct access to the Internet. If that succeeds, it will save the file in `DISTDIR` for future use and proceed.

2. The `extract` target is run. It looks for your port's distribution file (typically a `gzip`ped tarball) in `DISTDIR` and unpacks it into a temporary subdirectory specified by `WRKDIR` (defaults to `work`).

3. The `patch` target is run. First, any patches defined in `PATCHFILES` are applied. Second, if any patch files named `patch-*` are found in `PATCHDIR` (defaults to the `files` subdirectory), they are applied at this time in alphabetical order.

4. The `configure` target is run. This can do any one of many different things.

    1. If it exists, `scripts/configure` is run.

    2. If `HAS_CONFIGURE` or `GNU_CONFIGURE` is set, `WRKSRC/configure` is run.

5. The `build` target is run. This is responsible for descending into the port's private working directory (`WRKSRC`) and building it.

6. The `stage` target is run. This puts the final set of built files into a temporary directory (`STAGEDIR`, see Section 6.1, “Staging”). The hierarchy of this directory mirrors that of the system on which the package will be installed.

7. The `install` target is run. This copies the files listed in the port's pkg-plist to the host system.

The above are the default actions. In addition, you can define targets `pre-`*`something`* or `post-`*`something`*, or put scripts with those names, in the `scripts` subdirectory, and they will be run before or after the default actions are done.

For example, if you have a `post-extract` target defined in your `Makefile`, and a file `pre-build` in the `scripts` subdirectory, the `post-extract` target will be called after the regular extraction actions, and the `pre-build` script will be executed before the default build rules are done. It is recommended that you use `Makefile` targets if the actions are simple enough, because it will be easier for someone to figure out what kind of non-default action the port requires.

The default actions are done by the `bsd.port.mk` targets `do-`*`something`*. For example, the commands to extract a port are in the target `do-extract`. If you are not happy with the default target, you can fix it by redefining the `do-`*`something`* target in your `Makefile`.

Note

The "main" targets (e.g., `extract`, `configure`, etc.) do nothing more than make sure all the stages up to that one are completed and call the real targets or scripts, and they are not intended to be changed. If you want to fix the extraction, fix `do-extract`, but never ever change the way `extract` operates! Additionally, the target `post-deinstall` is invalid and is not run by the ports infrastructure.

Now that you understand what goes on when the user types `make install`, let us go through the recommended steps to create the perfect port.

4.2. Getting the Original Sources

Get the original sources (normally) as a compressed tarball (`foo.tar.gz` or `foo.tar.bz2`) and copy it into `DISTDIR`. Always use *mainstream* sources when and where you can.

You will need to set the variable `MASTER_SITES` to reflect where the original tarball resides. You will find convenient shorthand definitions for most mainstream sites in `bsd.sites.mk`. Please use these sites—and the associated definitions—if at all possible, to help avoid the problem of having the same information repeated over again many times in the source base. As these sites tend to change over time, this becomes a maintenance nightmare for everyone involved.

If you cannot find a FTP/HTTP site that is well-connected to the net, or can only find sites that have irritatingly non-standard formats, you might want to put a copy on a reliable FTP or HTTP server that you control (e.g., your home page).

If you cannot find somewhere convenient and reliable to put the distfile we can “house” it ourselves on `ftp.FreeBSD.org` ; however, this is the least-preferred solution. The distfile must be placed into `~/public_distfiles/` of someone's `freefall` account. Ask the person who commits your port to do this. This person will also set `MASTER_SITES` to `MASTER_SITE_LOCAL` and `MASTER_SITE_SUBDIR` to their `freefall` username.

If your port's distfile changes all the time without any kind of version update by the author, consider putting the distfile on your home page and listing it as the first `MASTER_SITES` . If you can, try to talk the port author out of doing this; it really does help to establish some kind of source code control. Hosting your own version will prevent users from getting checksum mismatch errors, and also reduce the workload of maintainers of our FTP site. Also, if there is only one master site for the port, it is recommended that you house a backup at your site and list it as the second `MASTER_SITES` .

If your port requires some additional `patches' that are available on the Internet, fetch them too and put them in `DISTDIR`. Do not worry if they come from a site other than where you got the main source tarball, we have a way to handle these situations (see the description of PATCHFILES below).

4.3. Modifying the Port

Unpack a copy of the tarball in a private directory and make whatever changes are necessary to get the port to compile properly under the current version of FreeBSD. Keep *careful track* of everything you do, as you will be automating the process shortly. Everything, including the deletion, addition, or modification of files should be doable using an automated script or patch file when your port is finished.

If your port requires significant user interaction/customization to compile or install, you should take a look at one of Larry Wall's classic Configure scripts and perhaps do something similar yourself. The goal of the new ports collection is to make each port as “plug-and-play” as possible for the end-user while using a minimum of disk space.

Note

Unless explicitly stated, patch files, scripts, and other files you have created and contributed to the FreeBSD ports collection are assumed to be covered by the standard BSD copyright conditions.

4.4. Patching

In the preparation of the port, files that have been added or changed can be picked up with a diff(1) for later feeding to patch(1). Each patch you wish to apply should be saved into a file named `patch-*` where * indicates the pathname of the file that is patched, such as `patch-Imakefile` or `patch-src-config.h` . These files should be stored in `PATCHDIR` (usually `files/`, from where they will be automatically applied. All patches must be relative to `WRKSRC` (generally the directory your port's tarball unpacks itself into, that being where the build is done). To make fixes and upgrades easier, you should avoid having more than one patch fix the same file (e.g., `patch-file` and `patch-file2` both changing `WRKSRC/foobar.c`). Note that if the path of a patched file contains an underscore (_) character, the patch needs to have two underscores instead in its name. For example, to patch a file named `src/freeglut_joystick.c` , the corresponding patch should be named `patch-src-freeglut__joystick.c` .

Please only use characters `[-+._a-zA-Z0-9]` for naming your patches. Do not use any other characters besides them. Do not name your patches like `patch-aa` or `patch-ab` etc, always mention the path and file name in patch names.

Do not put RCS strings in patches. SVN will mangle them when we put the files into the ports tree, and when we check them out again, they will come out different and the patch will fail. RCS strings are surrounded by dollar (`$`) signs, and typically start with `$Id` or `$RCS`.

Using the recurse (`-r`) option to diff(1) to generate patches is fine, but please take a look at the resulting patches to make sure you do not have any unnecessary junk in there. In particular, diffs between two backup files, `Makefile`s when the port uses `Imake` or GNU `configure`, etc., are unnecessary and should be deleted. If you had to edit `configure.in` and run `autoconf` to regenerate `configure`, do not take the diffs of `configure` (it often grows to a few thousand lines!); define `USE_AUTOTOOLS=autoconf:261` and take the diffs of `configure.in` .

Also, try to minimize the amount of non-functional whitespace changes in your patches. It is common in the Open Source world for projects to share large amounts of a code base, but obey different style and indenting rules. If you take a working piece of functionality from one project to fix similar areas in another, please be careful: the resulting line patch may be full of non-functional changes. It not only increases the size of the SVN repository but makes it hard to find out what exactly caused the problem and what you changed at all.

If you had to delete a file, then you can do it in the `post-extract` target rather than as part of the patch.

Simple replacements can be performed directly from the port `Makefile` using the in-place mode of sed(1). This is very useful when you need to patch in a variable value. Example:

```
post-patch:
```

```
@${REINPLACE_CMD} --e -'s|for Linux|for FreeBSD|g' ${WRKSRC}/README
```

Quite often, software being ported uses the CR/LF convention for its source files. This may cause problems with further patching, compiler warnings, scripts execution (e.g., `/bin/sh^M` not found.) To quickly convert all files from CR/LF to just LF, add `USES=dos2unix` to the port `Makefile`. A list of files to convert can be specified:

```
DOS2UNIX_FILES= util.c util.h
```

If you want to convert a group of files across subdirectories, `DOS2UNIX_REGEX` can be used. Its argument is a `find` compatible regular expression. More on the format is in re_format(7). This option is useful for converting all files of a given extension, for example all source code files leaving binary files intact:

```
USES= dos2unix
DOS2UNIX_REGEX= .*\.([ch]|cpp)
```

A similar option is `DOS2UNIX_GLOB`, which invokes `find` for each element listed in it.

```
USES= dos2unix
DOS2UNIX_GLOB= *.c *.cpp *.h
```

If you want to create a patch file based off of an existing file, you can copy it with an `.orig` extension, and then modify the original one. The `makepatch` target will write out an appropriate patch file to the `files` directory of the port.

4.5. Configuring

Include any additional customization commands in your `configure` script and save it in the `scripts` subdirectory. As mentioned above, you can also do this with `Makefile` targets and/or scripts with the name `pre-configure` or `post-configure`.

4.6. Handling User Input

If your port requires user input to build, configure, or install, you must set `IS_INTERACTIVE` in your `Makefile`. This will allow "overnight builds" to skip your port if the user sets the variable `BATCH` in his environment (and if the user sets the variable `INTERACTIVE`, then *only* those ports requiring interaction are built). This will save a lot of wasted time on the set of machines that continually build ports (see below).

It is also recommended that if there are reasonable default answers to the questions, you check the `PACKAGE_BUILDING` variable and turn off the interactive script when it is set. This will allow us to build the packages for CDROMs and FTP.

Chapter 5. Configuring the Makefile

Configuring the `Makefile` is pretty simple, and again we suggest that you look at existing examples before starting. Also, there is a sample Makefile in this handbook, so take a look and please follow the ordering of variables and sections in that template to make your port easier for others to read.

Now, consider the following problems in sequence as you design your new `Makefile`:

5.1. The Original Source

Does it live in `DISTDIR` as a standard `gzipped` tarball named something like `foozolix-1.2.tar.gz` ? If so, you can go on to the next step. If not, you should look at overriding any of the `DISTVERSION` , `DISTNAME`, `EXTRACT_CMD` , `EXTRACT_BEFORE_ARGS` , `EXTRACT_AFTER_ARGS` , `EXTRACT_SUFX` , or `DISTFILES` variables, depending on how alien a format your port's distribution file is.

In the worst case, you can simply create your own `do-extract` target to override the default, though this should be rarely, if ever, necessary.

5.2. Naming

The first part of the port's `Makefile` names the port, describes its version number, and lists it in the correct category.

5.2.1. `PORTNAME` and `PORTVERSION`

You should set `PORTNAME` to the base name of your port, and `PORTVERSION` to the version number of the port.

5.2.2. `PORTREVISION` and `PORTEPOCH`

5.2.2.1. `PORTREVISION`

The `PORTREVISION` variable is a monotonically increasing value which is reset to `0` with every increase of `PORTVERSION` (i.e., every time a new official vendor release is made), and appended to the package name if non-zero. Changes to `PORTREVISION` are used by automated tools (e.g., pkg_version(1)) to highlight the fact that a new package is available.

`PORTREVISION` should be increased each time a change is made to the port which significantly affects the content or structure of the derived package.

Examples of when PORTREVISION should be bumped:

- Addition of patches to correct security vulnerabilities, bugs, or to add new functionality to the port.
- Changes to the port Makefile to enable or disable compile-time options in the package.
- Changes in the packing list or the install-time behavior of the package (e.g., change to a script which generates initial data for the package, like ssh host keys).
- Version bump of a port's shared library dependency (in this case, someone trying to install the old package after installing a newer version of the dependency will fail since it will look for the old libfoo.x instead of libfoo.(x+1)).
- Silent changes to the port distfile which have significant functional differences, i.e., changes to the distfile requiring a correction to distinfo with no corresponding change to PORTVERSION , where a diff -ru of the old and new versions shows non-trivial changes to the code.

Examples of changes which do not require a PORTREVISION bump:

- Style changes to the port skeleton with no functional change to what appears in the resulting package.
- Changes to MASTER_SITES or other functional changes to the port which do not affect the resulting package.
- Trivial patches to the distfile such as correction of typos, which are not important enough that users of the package should go to the trouble of upgrading.
- Build fixes which cause a package to become compilable where it was previously failing (as long as the changes do not introduce any functional change on any other platforms on which the port did previously build). Since PORTREVISION reflects the content of the package, if the package was not previously buildable then there is no need to increase PORTREVISION to mark a change.

A rule of thumb is to ask yourself whether a change committed to a port is something which everyone would benefit from having (either because of an enhancement, fix, or by virtue that the new package will actually work at all), and weigh that against that fact that it will cause everyone who regularly updates their ports tree to be compelled to update. If yes, the PORTREVISION should be bumped.

5.2.2.2. PORTEPOCH

From time to time a software vendor or FreeBSD porter will do something silly and release a version of their software which is actually numerically less than the previous version. An example of this is a port which goes from foo-20000801 to foo-1.0 (the former will be

incorrectly treated as a newer version since 20000801 is a numerically greater value than 1).

Tip

The results of version number comparisons are not always obvious. pkg_version(1) can be used to test the comparison of two version number strings. The pkgng equivalent is `pkg version -t`. For example:

```
% pkg_version --t 0.031 0.29
>
```

Or, for pkgng users:

```
% pkg version --t 0.031 0.29
>
```

The > output indicates that version 0.031 is considered greater than version 0.29, which may not have been obvious to the porter.

In situations such as this, the `PORTEPOCH` version should be increased. If `PORTEPOCH` is nonzero it is appended to the package name as described in section 0 above. `PORTEPOCH` must never be decreased or reset to zero, because that would cause comparison to a package from an earlier epoch to fail (i.e., the package would not be detected as out of date): the new version number (e.g., `1.0,1` in the above example) is still numerically less than the previous version (20000801), but the `,1` suffix is treated specially by automated tools and found to be greater than the implied suffix `,0` on the earlier package.

Dropping or resetting `PORTEPOCH` incorrectly leads to no end of grief; if you do not understand the above discussion, please keep after it until you do, or ask questions on the mailing lists.

It is expected that `PORTEPOCH` will not be used for the majority of ports, and that sensible use of `PORTVERSION` can often preempt it becoming necessary if a future release of the software should change the version structure. However, care is needed by FreeBSD porters when a vendor release is made without an official version number — such as a code "snapshot" release. The temptation is to label the release with the release date, which will cause problems as in the example above when a new "official" release is made.

For example, if a snapshot release is made on the date 20000917, and the previous version of the software was version 1.2, the snapshot release should be given a `PORTVERSION` of 1.2.20000917 or similar, not 20000917, so that the succeeding release, say 1.3, is still a numerically greater value.

5.2.2.3. Example of PORTREVISION and PORTEPOCH Usage

The gtkmumble port, version 0.10, is committed to the ports collection:

```
PORTNAME= gtkmumble
PORTVERSION= 0.10
```

PKGNAME becomes gtkmumble-0.10.

A security hole is discovered which requires a local FreeBSD patch. PORTREVISION is bumped accordingly.

```
PORTNAME= gtkmumble
PORTVERSION= 0.10
PORTREVISION= 1
```

PKGNAME becomes gtkmumble-0.10_1

A new version is released by the vendor, numbered 0.2 (it turns out the author actually intended 0.10 to actually mean 0.1.0, not "what comes after 0.9" - oops, too late now). Since the new minor version 2 is numerically less than the previous version 10, the PORTEPOCH must be bumped to manually force the new package to be detected as "newer". Since it is a new vendor release of the code, PORTREVISION is reset to 0 (or removed from the Makefile).

```
PORTNAME= gtkmumble
PORTVERSION= 0.2
PORTEPOCH= 1
```

PKGNAME becomes gtkmumble-0.2,1

The next release is 0.3. Since PORTEPOCH never decreases, the version variables are now:

```
PORTNAME= gtkmumble
PORTVERSION= 0.3
PORTEPOCH= 1
```

PKGNAME becomes gtkmumble-0.3,1

Note

If PORTEPOCH were reset to 0 with this upgrade, someone who had installed the gtkmumble-0.10_1 package would not detect the gtkmumble-0.3 package as newer, since 3 is still numerically less than 10. Remember, this is the whole point of PORTEPOCH in the first place.

5.2.3. PKGNAMEPREFIX and PKGNAMESUFFIX

Two optional variables, PKGNAMEPREFIX and PKGNAMESUFFIX, are combined with PORTNAME and PORTVERSION to form PKGNAME as ${PKGNAMEPREFIX}${PORTNAME}${PKGNAMESUFFIX}-${PORTVERSION} . Make sure this conforms to our guidelines for a good package name. In particular, you are *not* allowed to use a hyphen (-) in PORTVERSION . Also, if the package name has the *language-* or the *-compiled.specifics* part (see below), use PKGNAMEPREFIX and PKGNAMESUFFIX, respectively. Do not make them part of PORTNAME.

5.2.4. LATEST_LINK

LATEST_LINK is used during package building to determine a shortened name to create links that can be used by pkg_add -r . This makes it possible to, for example, install the latest perl version by running pkg_add -r perl without knowing the exact version number. This name needs to be unique and obvious to users.

In some cases, several versions of a program may be present in the ports collection at the same time. Both the index build and the package build system need to be able to see them as different, independent ports, although they may all have the same PORTNAME, PKGNAMEPREFIX, and even PKGNAMESUFFIX. In those cases, the optional LATEST_LINK variable should be set to a different value for all ports except the "main" one — see the lang/gcc46 and lang/gcc ports, and the www/apache* family for examples of its use. By setting NO_LATEST_LINK , no link will be generated, which may be an option for all but the "main" version. Note that how to choose a "main" version — "most popular", "best supported", "least patched", and so on — is outside the scope of this handbook's recommendations; we only tell you how to specify the other ports' versions after you have picked a "main" one.

5.2.5. Package Naming Conventions

The following are the conventions you should follow in naming your packages. This is to have our package directory easy to scan, as there are already thousands of packages and users are going to turn away if they hurt their eyes!

The package name should look like language_region-name-compiled.specifics-version.numbers .

The package name is defined as ${PKGNAMEPREFIX}${PORTNAME}${PKGNAMESUFFIX}-${PORTVERSION} . Make sure to set the variables to conform to that format.

1. FreeBSD strives to support the native language of its users. The *language-* part should be a two letter abbreviation of the natural language defined by ISO-639 if the port is specific to a certain language. Examples are ja for Japanese, ru for Russian, vi for Vietnamese, zh for Chinese, ko for Korean and de for German.

   If the port is specific to a certain region within the language area, add the two letter country code as well. Examples are en_US for US English and fr_CH for Swiss French.

The *language-* part should be set in the PKGNAMEPREFIX variable.

2. The first letter of the name part should be lowercase. (The rest of the name may contain capital letters, so use your own discretion when you are converting a software name that has some capital letters in it.) There is a tradition of naming Perl 5 modules by prepending p5- and converting the double-colon separator to a hyphen; for example, the Data::Dumper module becomes p5-Data-Dumper.

3. Make sure that the port's name and version are clearly separated and placed into the PORTNAME and PORTVERSION variables. The only reason for PORTNAME to contain a version part is if the upstream distribution is really named that way, as in the textproc/libxml2 or japanese/kinput2-freewnn ports. Otherwise, the PORTNAME should not contain any version-specific information. It is quite normal for several ports to have the same PORTNAME, as the www/apache* ports do; in that case, different versions (and different index entries) are distinguished by the PKGNAMEPREFIX, PKGNAMESUFFIX, and LATEST_LINK values.

4. If the port can be built with different hardcoded defaults (usually part of the directory name in a family of ports), the *-compiled.specifics* part should state the compiled-in defaults (the hyphen is optional). Examples are paper size and font units.

   The *-compiled.specifics* part should be set in the PKGNAMESUFFIX variable.

5. The version string should follow a dash (-) and be a period-separated list of integers and single lowercase alphabetics. In particular, it is not permissible to have another dash inside the version string. The only exception is the string pl (meaning "patch-level"), which can be used *only* when there are no major and minor version numbers in the software. If the software version has strings like "alpha", "beta", "rc", or "pre", take the first letter and put it immediately after a period. If the version string continues after those names, the numbers should follow the single alphabet without an extra period between them.

   The idea is to make it easier to sort ports by looking at the version string. In particular, make sure version number components are always delimited by a period, and if the date is part of the string, use the 0.0.yyyy.mm.dd format, not dd.mm.yyyy or the non-Y2K compliant yy.mm.dd format. It is important to prefix the version with 0.0. in case a release with an actual version number is made, which would of course be numerically less than yyyy.

Here are some (real) examples on how to convert the name as called by the software authors to a suitable package name:

Distribution Name	**PKGNAMEPREFIX**	**PORTNAME**	**PKGNAMESUFFIX**	**PORTVERSION**	Reason
mule-2.2.2	(empty)	mule	(empty)	2.2.2	No changes required
Emi-Clock-1.0.2	(empty)	emiclock	(empty)	1.0.2	No uppercase names for single programs
rdist-1.3alpha	(empty)	rdist	(empty)	1.3.a	No strings like `alpha` allowed
es-0.9-beta1	(empty)	es	(empty)	0.9.b1	No strings like `beta` allowed
mailman-2.0rc3	(empty)	mailman	(empty)	2.0.r3	No strings like `rc` allowed
v3.3beta021.src	(empty)	tiff	(empty)	3.3	What the heck was that anyway?
tvtwm	(empty)	tvtwm	(empty)	pl11	Version string always required
piewm	(empty)	piewm	(empty)	1.0	Version string always required
xvgr-2.10pl1	(empty)	xvgr	(empty)	2.10.1	`pl` allowed only when no major/minor version numbers
gawk-2.15.6	ja-	gawk	(empty)	2.15.6	Japanese language version
psutils-1.13	(empty)	psutils	-letter	1.13	Paper size hardcoded at package build time

Distribution Name	PKGNAMEPREFIX	PORTNAME	PKGNAMESUFFIX	PORTVERSION	Reason
pkfonts	(empty)	pkfonts	300	1.0	Package for 300dpi fonts

If there is absolutely no trace of version information in the original source and it is unlikely that the original author will ever release another version, just set the version string to `1.0` (like the `piewm` example above). Otherwise, ask the original author or use the date string (`0.0.yyyy.mm.dd`) as the version.

5.3. Categorization

5.3.1. CATEGORIES

When a package is created, it is put under `/usr/ports/packages/All` and links are made from one or more subdirectories of `/usr/ports/packages` . The names of these subdirectories are specified by the variable `CATEGORIES`. It is intended to make life easier for the user when he is wading through the pile of packages on the FTP site or the CDROM. Please take a look at the current list of categories and pick the ones that are suitable for your port.

This list also determines where in the ports tree the port is imported. If you put more than one category here, it is assumed that the port files will be put in the subdirectory with the name in the first category. See below for more discussion about how to pick the right categories.

5.3.2. Current List of Categories

Here is the current list of port categories. Those marked with an asterisk (*) are *virtual* categories—those that do not have a corresponding subdirectory in the ports tree. They are only used as secondary categories, and only for search purposes.

Note

For non-virtual categories, you will find a one-line description in the `COMMENT` in that subdirectory's `Makefile`.

Category	Description	Notes
`accessibility`	Ports to help disabled users.	
`afterstep*`	Ports to support the AfterStep window manager.	

Category	Description	Notes
`arabic`	Arabic language support.	
`archivers`	Archiving tools.	
`astro`	Astronomical ports.	
`audio`	Sound support.	
`benchmarks`	Benchmarking utilities.	
`biology`	Biology-related software.	
`cad`	Computer aided design tools.	
`chinese`	Chinese language support.	
`comms`	Communication software.	Mostly software to talk to your serial port.
`converters`	Character code converters.	
`databases`	Databases.	
`deskutils`	Things that used to be on the desktop before computers were invented.	
`devel`	Development utilities.	Do not put libraries here just because they are libraries—unless they truly do not belong anywhere else, they should not be in this category.
`dns`	DNS-related software.	
`docs*`	Meta-ports for FreeBSD documentation.	
`editors`	General editors.	Specialized editors go in the section for those tools (e.g., a mathematical-formula editor will go in `math`).
`elisp*`	Emacs-lisp ports.	
`emulators`	Emulators for other operating systems.	Terminal emulators do *not* belong here—X-based ones should go to `x11` and text-based ones to either `comms` or `misc`, depending on the exact functionality.

Category	Description	Notes
finance	Monetary, financial and related applications.	
french	French language support.	
ftp	FTP client and server utilities.	If your port speaks both FTP and HTTP, put it in ftp with a secondary category of www.
games	Games.	
geography*	Geography-related software.	
german	German language support.	
gnome*	Ports from the GNOME Project.	
gnustep*	Software related to the GNUstep desktop environment.	
graphics	Graphics utilities.	
hamradio*	Software for amateur radio.	
haskell*	Software related to the Haskell language.	
hebrew	Hebrew language support.	
hungarian	Hungarian language support.	
ipv6*	IPv6 related software.	
irc	Internet Relay Chat utilities.	
japanese	Japanese language support.	
java	Software related to the Java™ language.	The java category must not be the only one for a port. Save for ports directly related to the Java language, porters are also encouraged not to use java as the main category of a port.
kde*	Ports from the KDE Project.	
kld*	Kernel loadable modules.	
korean	Korean language support.	

Category	Description	Notes
`lang`	Programming languages.	
`linux*`	Linux applications and support utilities.	
`lisp*`	Software related to the Lisp language.	
`mail`	Mail software.	
`math`	Numerical computation software and other utilities for mathematics.	
`mbone*`	MBone applications.	
`misc`	Miscellaneous utilities	Basically things that do not belong anywhere else. If at all possible, try to find a better category for your port than `misc`, as ports tend to get overlooked in here.
`multimedia`	Multimedia software.	
`net`	Miscellaneous networking software.	
`net-im`	Instant messaging software.	
`net-mgmt`	Networking management software.	
`net-p2p`	Peer to peer network applications.	
`news`	USENET news software.	
`palm`	Software support for the Palm™ series.	
`parallel*`	Applications dealing with parallelism in computing.	
`pear*`	Ports related to the Pear PHP framework.	
`perl5*`	Ports that require Perl version 5 to run.	
`plan9*`	Various programs from Plan9.	
`polish`	Polish language support.	

Category	Description	Notes
`ports-mgmt`	Ports for managing, installing and developing FreeBSD ports and packages.	
`portuguese`	Portuguese language support.	
`print`	Printing software.	Desktop publishing tools (previewers, etc.) belong here too.
`python*`	Software related to the Python language.	
`ruby*`	Software related to the Ruby language.	
`rubygems*`	Ports of RubyGems packages.	
`russian`	Russian language support.	
`scheme*`	Software related to the Scheme language.	
`science`	Scientific ports that do not fit into other categories such as `astro`, `biology` and `math`.	
`security`	Security utilities.	
`shells`	Command line shells.	
`spanish*`	Spanish language support.	
`sysutils`	System utilities.	
`tcl*`	Ports that use Tcl to run.	
`textproc`	Text processing utilities.	It does not include desktop publishing tools, which go to `print`.
`tk*`	Ports that use Tk to run.	
`ukrainian`	Ukrainian language support.	
`vietnamese`	Vietnamese language support.	
`windowmaker*`	Ports to support the WindowMaker window manager.	

Category	Description	Notes
`www`	Software related to the World Wide Web.	HTML language support belongs here too.
`x11`	The X Window System and friends.	This category is only for software that directly supports the window system. Do not put regular X applications here; most of them should go into other `x11-*` categories (see below).
`x11-clocks`	X11 clocks.	
`x11-drivers`	X11 drivers.	
`x11-fm`	X11 file managers.	
`x11-fonts`	X11 fonts and font utilities.	
`x11-servers`	X11 servers.	
`x11-themes`	X11 themes.	
`x11-toolkits`	X11 toolkits.	
`x11-wm`	X11 window managers.	
`xfce*`	Ports related to the Xfce desktop environment.	
`zope*`	Zope support.	

5.3.3. Choosing the Right Category

As many of the categories overlap, you often have to choose which of the categories should be the primary category of your port. There are several rules that govern this issue. Here is the list of priorities, in decreasing order of precedence:

- The first category must be a physical category (see above). This is necessary to make the packaging work. Virtual categories and physical categories may be intermixed after that.

- Language specific categories always come first. For example, if your port installs Japanese X11 fonts, then your `CATEGORIES` line would read `japanese x11-fonts`.

- Specific categories are listed before less-specific ones. For instance, an HTML editor should be listed as `www editors`, not the other way around. Also, you should not list `net` when the port belongs to any of `irc`, `mail`, `news`, `security`, or `www`, as `net` is included implicitly.

- `x11` is used as a secondary category only when the primary category is a natural language. In particular, you should not put `x11` in the category line for X applications.
- Emacs modes should be placed in the same ports category as the application supported by the mode, not in `editors`. For example, an Emacs mode to edit source files of some programming language should go into `lang`.
- Ports which install loadable kernel modules should have the virtual category `kld` in their `CATEGORIES` line. This is one of the things handled automatically by adding `kmod` to the `USES` line.
- `misc` should not appear with any other non-virtual category. If you have `misc` with something else in your `CATEGORIES` line, that means you can safely delete `misc` and just put the port in that other subdirectory!
- If your port truly does not belong anywhere else, put it in `misc`.

If you are not sure about the category, please put a comment to that effect in your send-pr(1) submission so we can discuss it before we import it. If you are a committer, send a note to the FreeBSD ports mailing list so we can discuss it first. Too often, new ports are imported to the wrong category only to be moved right away. This causes unnecessary and undesirable bloat in the master source repository.

5.3.4. Proposing a New Category

As the Ports Collection has grown over time, various new categories have been introduced. New categories can either be *virtual* categories—those that do not have a corresponding subdirectory in the ports tree— or *physical* categories—those that do. The following text discusses the issues involved in creating a new physical category so that you can understand them before you propose one.

Our existing practice has been to avoid creating a new physical category unless either a large number of ports would logically belong to it, or the ports that would belong to it are a logically distinct group that is of limited general interest (for instance, categories related to spoken human languages), or preferably both.

The rationale for this is that such a change creates a fair amount of work for both the committers and also for all users who track changes to the Ports Collection. In addition, proposed category changes just naturally seem to attract controversy. (Perhaps this is because there is no clear consensus on when a category is “too big”, nor whether categories should lend themselves to browsing (and thus what number of categories would be an ideal number), and so forth.)

Here is the procedure:

1. Propose the new category on FreeBSD ports mailing list. You should include a detailed rationale for the new category, including why you feel the existing categories are not sufficient, and the list of existing ports proposed to move. (If there are new

ports pending in GNATS that would fit this category, list them too.) If you are the maintainer and/or submitter, respectively, mention that as it may help you to make your case.

2. Participate in the discussion.

3. If it seems that there is support for your idea, file a PR which includes both the rationale and the list of existing ports that need to be moved. Ideally, this PR should also include patches for the following:

   - `Makefiles` for the new ports once they are repocopied
   - `Makefile` for the new category
   - `Makefile` for the old ports' categories
   - `Makefiles` for ports that depend on the old ports
   - (for extra credit, you can include the other files that have to change, as per the procedure in the Committer's Guide.)

4. Since it affects the ports infrastructure and involves not only performing repo-copies but also possibly running regression tests on the build cluster, the PR should be assigned to the Ports Management Team <portmgr@FreeBSD.org >.

5. If that PR is approved, a committer will need to follow the rest of the procedure that is outlined in the Committer's Guide.

Proposing a new virtual category should be similar to the above but much less involved, since no ports will actually have to move. In this case, the only patches to include in the PR would be those to add the new category to the `CATEGORIES` of the affected ports.

5.3.5. Proposing Reorganizing All the Categories

Occasionally someone proposes reorganizing the categories with either a 2-level structure, or some other kind of keyword structure. To date, nothing has come of any of these proposals because, while they are very easy to make, the effort involved to retrofit the entire existing ports collection with any kind of reorganization is daunting to say the very least. Please read the history of these proposals in the mailing list archives before you post this idea; furthermore, you should be prepared to be challenged to offer a working prototype.

5.4. The Distribution Files

The second part of the `Makefile` describes the files that must be downloaded in order to build the port, and where they can be downloaded from.

5.4.1. DISTVERSION/DISTNAME

DISTNAME is the name of the port as called by the authors of the software. DISTNAME defaults to ${PORTNAME}-${PORTVERSION} , so override it only if necessary. DISTNAME is only used in two places. First, the distribution file list (DISTFILES) defaults to ${DISTNAME} ${EXTRACT_SUFX} . Second, the distribution file is expected to extract into a subdirectory named WRKSRC , which defaults to work/${DISTNAME} .

Some vendor's distribution names which do not fit into the ${PORTNAME}-${PORTVERSION} -scheme can be handled automatically by setting DISTVERSION . PORTVERSION and DISTNAME will be derived automatically, but can of course be overridden. The following table lists some examples:

DISTVERSION	PORTVERSION
0.7.1d	0.7.1.d
10Alpha3	10.a3
3Beta7-pre2	3.b7.p2
8:f_17	8f.17

Note

PKGNAMEPREFIX and PKGNAMESUFFIX do not affect DISTNAME. Also note that if WRKSRC is equal to work/${PORTNAME}-${PORTVERSION} while the original source archive is named something other than ${PORTNAME}-${PORTVERSION}${EXTRACT_SUFX} , you should probably leave DISTNAME alone— you are better off defining DISTFILES than having to set both DISTNAME and WRKSRC (and possibly EXTRACT_SUFX).

5.4.2. MASTER_SITES

Record the directory part of the FTP/HTTP-URL pointing at the original tarball in MASTER_SITES . Do not forget the trailing slash (/)!

The make macros will try to use this specification for grabbing the distribution file with FETCH if they cannot find it already on the system.

It is recommended that you put multiple sites on this list, preferably from different continents. This will safeguard against wide-area network problems. We are even planning to add support for automatically determining the closest master site and fetching from there; having multiple sites will go a long way towards helping this effort.

If the original tarball is part of one of the popular archives such as SourceForge, GNU, or Perl CPAN, you may be able refer to those sites in an easy compact form using `MASTER_SITE_*` (e.g., `MASTER_SITE_SOURCEFORGE`, `MASTER_SITE_GNU` and `MASTER_SITE_PERL_CPAN`). Simply set `MASTER_SITES` to one of these variables and `MASTER_SITE_SUBDIR` to the path within the archive. Here is an example:

```
MASTER_SITES=  ${MASTER_SITE_GNU}
MASTER_SITE_SUBDIR= make
```

Or you can use a condensed format:

```
MASTER_SITES= GNU/make
```

These variables are defined in `/usr/ports/Mk/bsd.sites.mk`. There are new entries added all the time, so make sure to check the latest version of this file before submitting a port.

Several *magic* macros exist for popular sites with a predictable directory structure. For these, just use the abbreviation and the system will try to guess the correct subdirectory for you.

```
MASTER_SITES= SF
```

If the guess is incorrect, it can be overridden as follows.

```
MASTER_SITES= SF/stardict/WyabdcRealPeopleTTS/${PORTVERSION}
```

This can be also written as

```
MASTER_SITES= SF
MASTER_SITE_SUBDIR= stardict/WyabdcRealPeopleTTS/${PORTVERSION}
```

Table 5.1. Popular Magic `MASTER_SITES` Macros

Macro	Assumed subdirectory
APACHE_JAKARTA	/dist/jakarta/${PORTNAME:S,-,,/,}/source
BERLIOS	/${PORTNAME:L}
CHEESESHOP	/packages/source/source/${DISTNAME:C/(.).*/\1/}/${DISTNAME:C/(.*)-[0-9].*/\1/}
DEBIAN	/debian/pool/main/${PORTNAME:C/^((lib)?.).*$/\1/}/${PORTNAME}
GCC	/pub/gcc/releases/${DISTNAME}
GNOME	/pub/GNOME/sources/${PORTNAME}/${PORTVERSION:C/^([0-9]+\.[0-9]+).*/\1/}

Macro	Assumed subdirectory
GNU	/gnu/${PORTNAME}
MOZDEV	/pub/mozdev/${PORTNAME:L}
PERL_CPAN	/pub/CPAN/modules/by-module/${PORTNAME:C/-.*//}
PYTHON	/ftp/python/${PYTHON_PORTVERSION:C/rc[0-9]//}
RUBYFORGE	/${PORTNAME:L}
SAVANNAH	/${PORTNAME:L}
SF	/project/${PORTNAME:L}/${PORTNAME:L}/${PORTVERSION}

5.4.3. EXTRACT_SUFX

If you have one distribution file, and it uses an odd suffix to indicate the compression mechanism, set EXTRACT_SUFX .

For example, if the distribution file was named foo.tgz instead of the more normal foo.tar.gz , you would write:

```
DISTNAME= foo
EXTRACT_SUFX= .tgz
```

The USE_BZIP2 , USE_XZ and USE_ZIP variables automatically set EXTRACT_SUFX to .tar.bz2 , .tar.xz or .zip as necessary. If neither of these are set then EXTRACT_SUFX defaults to .tar.gz .

Note

You never need to set both EXTRACT_SUFX and DISTFILES.

5.4.4. DISTFILES

Sometimes the names of the files to be downloaded have no resemblance to the name of the port. For example, it might be called source.tar.gz or similar. In other cases the application's source code might be in several different archives, all of which must be downloaded.

If this is the case, set DISTFILES to be a space separated list of all the files that must be downloaded.

```
DISTFILES= source1.tar.gz source2.tar.gz
```

If not explicitly set, DISTFILES defaults to ${DISTNAME}${EXTRACT_SUFX} .

5.4.5. EXTRACT_ONLY

If only some of the DISTFILES must be extracted—for example, one of them is the source code, while another is an uncompressed document—list the filenames that must be extracted in EXTRACT_ONLY .

```
DISTFILES= source.tar.gz manual.html
EXTRACT_ONLY= source.tar.gz
```

If *none* of the DISTFILES should be uncompressed then set EXTRACT_ONLY to the empty string.

```
EXTRACT_ONLY=
```

5.4.6. PATCHFILES

If your port requires some additional patches that are available by FTP or HTTP, set PATCHFILES to the names of the files and PATCH_SITES to the URL of the directory that contains them (the format is the same as MASTER_SITES).

If the patch is not relative to the top of the source tree (i.e., WRKSRC) because it contains some extra pathnames, set PATCH_DIST_STRIP accordingly. For instance, if all the pathnames in the patch have an extra foozolix-1.0/ in front of the filenames, then set PATCH_DIST_STRIP=-p1 .

Do not worry if the patches are compressed; they will be decompressed automatically if the filenames end with .gz or .Z.

If the patch is distributed with some other files, such as documentation, in a gzipped tarball, you cannot just use PATCHFILES. If that is the case, add the name and the location of the patch tarball to DISTFILES and MASTER_SITES . Then, use the EXTRA_PATCHES variable to point to those files and bsd.port.mk will automatically apply them for you. In particular, do *not* copy patch files into the PATCHDIR directory—that directory may not be writable.

Note

The tarball will have been extracted alongside the regular source by then, so there is no need to explicitly extract it if it is a regular gzipped or compressed tarball. If you do the latter, take extra care not to overwrite something that already exists in that directory.

Also, do not forget to add a command to remove the copied patch in the `pre-clean` target.

5.4.7. Multiple Distribution Files or Patches from Different Sites and Subdirectories (`MASTER_SITES:n`)

(Consider this to be a somewhat "advanced topic"; those new to this document may wish to skip this section at first).

This section has information on the fetching mechanism known as both `MASTER_SITES:n` and `MASTER_SITES_NN` . We will refer to this mechanism as `MASTER_SITES:n` .

A little background first. OpenBSD has a neat feature inside the `DISTFILES` and `PATCHFILES` variables which allows files and patches to be postfixed with `:n` identifiers. Here, `n` can be both `[0-9]` and denote a group designation. For example:

```
DISTFILES= alpha:0 beta:1
```

In OpenBSD, distribution file `alpha` will be associated with variable `MASTER_SITES0` instead of our common `MASTER_SITES` and `beta` with `MASTER_SITES1` .

This is a very interesting feature which can decrease that endless search for the correct download site.

Just picture 2 files in `DISTFILES` and 20 sites in `MASTER_SITES` , the sites slow as hell where `beta` is carried by all sites in `MASTER_SITES` , and `alpha` can only be found in the 20th site. It would be such a waste to check all of them if the maintainer knew this beforehand, would it not? Not a good start for that lovely weekend!

Now that you have the idea, just imagine more `DISTFILES` and more `MASTER_SITES` . Surely our "distfiles survey meister" would appreciate the relief to network strain that this would bring.

In the next sections, information will follow on the FreeBSD implementation of this idea. We improved a bit on OpenBSD's concept.

5.4.7.1. Simplified Information

This section tells you how to quickly prepare fine grained fetching of multiple distribution files and patches from different sites and subdirectories. We describe here a case of simplified `MASTER_SITES:n` usage. This will be sufficient for most scenarios. However, if you need further information, you will have to refer to the next section.

Some applications consist of multiple distribution files that must be downloaded from a number of different sites. For example, Ghostscript consists of the core of the program, and then a large number of driver files that are used depending on the user's printer. Some

of these driver files are supplied with the core, but many others must be downloaded from a variety of different sites.

To support this, each entry in DISTFILES may be followed by a colon and a "tag name". Each site listed in MASTER_SITES is then followed by a colon, and the tag that indicates which distribution files should be downloaded from this site.

For example, consider an application with the source split in two parts, source1.tar.gz and source2.tar.gz , which must be downloaded from two different sites. The port's Makefile would include lines like Example 5.1, "Simplified Use of MASTER_SITES:n with One File Per Site".

Example 5.1. Simplified Use of MASTER_SITES:n with One File Per Site

```
MASTER_SITES= ftp://ftp.example1.com/:source1 \
  ftp://ftp.example2.com/:source2
DISTFILES= source1.tar.gz:source1 \
  source2.tar.gz:source2
```

Multiple distribution files can have the same tag. Continuing the previous example, suppose that there was a third distfile, source3.tar.gz , that should be downloaded from ftp.example2.com . The Makefile would then be written like Example 5.2, "Simplified Use of MASTER_SITES:n with More Than One File Per Site".

Example 5.2. Simplified Use of MASTER_SITES:n with More Than One File Per Site

```
MASTER_SITES= ftp://ftp.example1.com/:source1 \
  ftp://ftp.example2.com/:source2
DISTFILES= source1.tar.gz:source1 \
  source2.tar.gz:source2 \
  source3.tar.gz:source2
```

5.4.7.2. Detailed Information

Okay, so the previous section example did not reflect your needs? In this section we will explain in detail how the fine grained fetching mechanism MASTER_SITES:n works and how you can modify your ports to use it.

1. Elements can be postfixed with `:n` where *n* is `[^:,]+`, i.e., *n* could conceptually be any alphanumeric string but we will limit it to `[a-zA-Z_][0-9a-zA-Z_]+` for now.

   Moreover, string matching is case sensitive; i.e., `n` is different from `N`.

   However, the following words cannot be used for postfixing purposes since they yield special meaning: `default`, `all` and `ALL` (they are used internally in item ii). Furthermore, `DEFAULT` is a special purpose word (check item 3).

2. Elements postfixed with `:n` belong to the group `n`, `:m` belong to group `m` and so forth.

3. Elements without a postfix are groupless, i.e., they all belong to the special group `DEFAULT`. If you postfix any elements with `DEFAULT`, you are just being redundant unless you want to have an element belonging to both `DEFAULT` and other groups at the same time (check item 5).

   The following examples are equivalent but the first one is preferred:

   ```
   MASTER_SITES= alpha
   ```

   ```
   MASTER_SITES= alpha:DEFAULT
   ```

4. Groups are not exclusive, an element may belong to several different groups at the same time and a group can either have either several different elements or none at all. Repeated elements within the same group will be simply that, repeated elements.

5. When you want an element to belong to several groups at the same time, you can use the comma operator (`,`).

   Instead of repeating it several times, each time with a different postfix, we can list several groups at once in a single postfix. For instance, `:m,n,o` marks an element that belongs to group `m`, `n` and `o`.

   All the following examples are equivalent but the last one is preferred:

   ```
   MASTER_SITES= alpha alpha:SOME_SITE
   ```

   ```
   MASTER_SITES= alpha:DEFAULT alpha:SOME_SITE
   ```

   ```
   MASTER_SITES= alpha:SOME_SITE,DEFAULT
   ```

   ```
   MASTER_SITES= alpha:DEFAULT,SOME_SITE
   ```

6. All sites within a given group are sorted according to `MASTER_SORT_AWK` . All groups within `MASTER_SITES` and `PATCH_SITES` are sorted as well.

7. Group semantics can be used in any of the following variables `MASTER_SITES` , `PATCH_SITES` , `MASTER_SITE_SUBDIR` , `PATCH_SITE_SUBDIR` , `DISTFILES`, and `PATCHFILES` according to the following syntax:

a. All MASTER_SITES , PATCH_SITES , MASTER_SITE_SUBDIR and PATCH_SITE_SUBDIR elements must be terminated with the forward slash / character. If any elements belong to any groups, the group postfix :n must come right after the terminator /. The MASTER_SITES:n mechanism relies on the existence of the terminator / to avoid confusing elements where a :n is a valid part of the element with occurrences where :n denotes group n. For compatibility purposes, since the / terminator was not required before in both MASTER_SITE_SUBDIR and PATCH_SITE_SUBDIR elements, if the postfix immediate preceding character is not a / then :n will be considered a valid part of the element instead of a group postfix even if an element is postfixed with :n. See both Example 5.3, "Detailed Use of MASTER_SITES:n in MASTER_SITE_SUBDIR " and Example 5.4, "Detailed Use of MASTER_SITES:n with Comma Operator, Multiple Files, Multiple Sites and Multiple Subdirectories".

Example 5.3. Detailed Use of MASTER_SITES:n in MASTER_SITE_SUBDIR

```
MASTER_SITE_SUBDIR= old:n new/:NEW
```

- Directories within group DEFAULT -> old:n
- Directories within group NEW -> new

Example 5.4. Detailed Use of MASTER_SITES:n with Comma Operator, Multiple Files, Multiple Sites and Multiple Subdirectories

```
MASTER_SITES= http://site1/%SUBDIR%/ http://
site2/:DEFAULT \
  http://site3/:group3 http://site4/:group4 \
  http://site5/:group5 http://site6/:group6 \
  http://site7/:DEFAULT,group6 \
  http://site8/%SUBDIR%/:group6,group7 \
  http://site9/:group8
DISTFILES= file1 file2:DEFAULT file3:group3 \
  file4:group4,group5,group6 file5:grouping \
  file6:group7
MASTER_SITE_SUBDIR= directory-trial:1 directory-
n/:groupn \
   directory-one/:group6,DEFAULT \
   directory
```

The previous example results in the following fine grained fetching. Sites are listed in the exact order they will be used.

- `file1` will be fetched from
  - `MASTER_SITE_OVERRIDE`
  - http://site1/directory-trial:1/
  - http://site1/directory-one/
  - http://site1/directory/
  - http://site2/
  - http://site7/
  - `MASTER_SITE_BACKUP`
- `file2` will be fetched exactly as `file1` since they both belong to the same group
  - `MASTER_SITE_OVERRIDE`
  - http://site1/directory-trial:1/
  - http://site1/directory-one/
  - http://site1/directory/
  - http://site2/
  - http://site7/
  - `MASTER_SITE_BACKUP`
- `file3` will be fetched from
  - `MASTER_SITE_OVERRIDE`
  - http://site3/
  - `MASTER_SITE_BACKUP`
- `file4` will be fetched from
  - `MASTER_SITE_OVERRIDE`

- http://site4/
- http://site5/
- http://site6/
- http://site7/
- http://site8/directory-one/
- MASTER_SITE_BACKUP

- file5 will be fetched from
  - MASTER_SITE_OVERRIDE
  - MASTER_SITE_BACKUP
- file6 will be fetched from
  - MASTER_SITE_OVERRIDE
  - http://site8/
  - MASTER_SITE_BACKUP

8. How do I group one of the special variables from bsd.sites.mk, e.g., MASTER_SITE_SOURCEFORGE ?

   See Example 5.5, "Detailed Use of MASTER_SITES:n with MASTER_SITE_SOURCEFORGE ".

Example 5.5. Detailed Use of MASTER_SITES:n with MASTER_SITE_SOURCEFORGE

```
MASTER_SITES= http://site1/ ${MASTER_SITE_SOURCEFORGE:S/
$/:sourceforge,TEST/}
DISTFILES= something.tar.gz:sourceforge
```

   something.tar.gz will be fetched from all sites within MASTER_SITE_SOURCEFORGE .

9. How do I use this with PATCH* variables?

All examples were done with MASTER* variables but they work exactly the same for PATCH* ones as can be seen in Example 5.6, "Simplified Use of MASTER_SITES:n with PATCH_SITES ".

Example 5.6. Simplified Use of MASTER_SITES:n with PATCH_SITES

```
PATCH_SITES= http://site1/ http://site2/:test
PATCHFILES= patch1:test
```

5.4.7.3. What Does Change for Ports? What Does Not?

i. All current ports remain the same. The MASTER_SITES:n feature code is only activated if there are elements postfixed with :n like elements according to the aforementioned syntax rules, especially as shown in item 7.

ii. The port targets remain the same: checksum, makesum, patch, configure, build, etc. With the obvious exceptions of do-fetch , fetch-list , master-sites and patch-sites.

- do-fetch : deploys the new grouping postfixed DISTFILES and PATCHFILES with their matching group elements within both MASTER_SITES and PATCH_SITES which use matching group elements within both MASTER_SITE_SUBDIR and PATCH_SITE_SUBDIR . Check Example 5.4, "Detailed Use of MASTER_SITES:n with Comma Operator, Multiple Files, Multiple Sites and Multiple Subdirectories".

- fetch-list : works like old fetch-list with the exception that it groups just like do-fetch .

- master-sites and patch-sites : (incompatible with older versions) only return the elements of group DEFAULT; in fact, they execute targets master-sites-default and patch-sites-default respectively.

  Furthermore, using target either master-sites-all or patch-sites-all is preferred to directly checking either MASTER_SITES or PATCH_SITES . Also, directly checking is not guaranteed to work in any future versions. Check item B for more information on these new port targets.

iii. New port targets

A. There are `master-sites-`*n* and `patch-sites-`*n* targets which will list the elements of the respective group *n* within `MASTER_SITES` and `PATCH_SITES` respectively. For instance, both `master-sites-DEFAULT` and `patch-sites-DEFAULT` will return the elements of group `DEFAULT`, `master-sites-test` and `patch-sites-test` of group `test`, and thereon.

B. There are new targets `master-sites-all` and `patch-sites-all` which do the work of the old `master-sites` and `patch-sites` ones. They return the elements of all groups as if they all belonged to the same group with the caveat that it lists as many `MASTER_SITE_BACKUP` and `MASTER_SITE_OVERRIDE` as there are groups defined within either `DISTFILES` or `PATCHFILES`; respectively for `master-sites-all` and `patch-sites-all` .

5.4.8. `DIST_SUBDIR`

Do not let your port clutter `/usr/ports/distfiles` . If your port requires a lot of files to be fetched, or contains a file that has a name that might conflict with other ports (e.g., `Makefile`), set `DIST_SUBDIR` to the name of the port (`${PORTNAME}` or `${PKGNAMEPREFIX}${PORTNAME}` should work fine). This will change `DISTDIR` from the default `/usr/ports/distfiles` to `/usr/ports/distfiles/DIST_SUBDIR` , and in effect puts everything that is required for your port into that subdirectory.

It will also look at the subdirectory with the same name on the backup master site at `ftp.FreeBSD.org` . (Setting `DISTDIR` explicitly in your `Makefile` will not accomplish this, so please use `DIST_SUBDIR` .)

Note

This does not affect the `MASTER_SITES` you define in your `Makefile`.

5.4.9. `ALWAYS_KEEP_DISTFILES`

If your port uses binary distfiles and has a license that requires that the source code is provided with packages distributed in binary form, e.g., GPL, `ALWAYS_KEEP_DISTFILES` will instruct the FreeBSD build cluster to keep a copy of the files specified in `DISTFILES`. Users of these ports will generally not need these files, so it is a good idea to only add the source distfiles to `DISTFILES` when `PACKAGE_BUILDING` is defined.

Example 5.7. Use of **ALWAYS_KEEP_DISTFILES**

```
.if defined(PACKAGE_BUILDING)
DISTFILES+=  foo.tar.gz
ALWAYS_KEEP_DISTFILES= yes
.endif
```

When adding extra files to DISTFILES, make sure you also add them to distinfo. Also, the additional files will normally be extracted into WRKDIR as well, which for some ports may lead to undesirable side effects and require special handling.

5.5. MAINTAINER

Set your mail-address here. Please. *:-)*

Note that only a single address without the comment part is allowed as a MAINTAINER value. The format used should be user@hostname.domain . Please do not include any descriptive text such as your real name in this entry—that merely confuses bsd.port.mk .

The maintainer is responsible for keeping the port up to date, and ensuring the port works correctly. For a detailed description of the responsibilities of a port maintainer, refer to the The challenge for port maintainers section.

Changes to the port will be sent to the maintainer of a port for review and approval before being committed. If the maintainer does not respond to an update request after two weeks (excluding major public holidays), then that is considered a maintainer timeout, and the update may be made without explicit maintainer approval. If the maintainer does not respond within three months, then that maintainer is considered absent without leave, and can be replaced as the maintainer of the particular port in question. Exceptions to this are anything maintained by the Ports Management Team <portmgr@FreeBSD.org >, or the Security Officer Team <security-officer@FreeBSD.org >. No unauthorized commits may ever be made to ports maintained by those groups.

We reserve the right to modify the maintainer's submission to better match existing policies and style of the Ports Collection without explicit blessing from the submitter. Also, large infrastructural changes can result in a port being modified without the maintainer's consent. These kinds of changes will never affect the port's functionality.

The Ports Management Team <portmgr@FreeBSD.org > reserves the right to revoke or override anyone's maintainership for any reason, and the Security Officer Team

<security-officer@FreeBSD.org> reserves the right to revoke or override maintainership for security reasons.

5.6. COMMENT

This is a one-line description of the port. Please respect the following rules:

1. Try to keep the COMMENT value at no longer than 70 characters, as this line will be used by the pkg_info(1) utility to display a one-line summary of the port;
2. Do *not* include the package name (or version number of the software);
3. The comment should begin with a capital and end without a period;
4. Do not start with an indefinite article (i.e., A or An);
5. Names are capitalized (for example, Apache, JavaScript, Perl);
6. For lists of words, use the Oxford comma (e.g., green, red, and blue);
7. Spell check the text.

Here is an example:

```
COMMENT= Cat chasing a mouse all over the screen
```

The COMMENT variable should immediately follow the MAINTAINER variable in the `Makefile`.

5.7. PORTSCOUT

Portscout is an automated distfile check utility for the FreeBSD Ports Collection, described in detail in Section 14.5, “Portscout: the FreeBSD Ports Distfile Scanner”.

The `PORTSCOUT` variable defines special conditions within which the Portscout distfile scanner should be restricted.

Situations where the `PORTSCOUT` variable should be set include:

- When distfiles should be ignored, whether for specific versions, or specific minor revisions. For example, to exclude version *`8.2`* from distfile version checks because it is known to be broken, add:

  ```
  PORTSCOUT= ignore:8.2
  ```

- When specific versions or specific major and minor revisions of a distfile should be checked. For example, if only version *`0.6.4`* should be monitored because newer versions have compatablity issues with FreeBSD, add:

```
PORTSCOUT= limit:^0\.6\.4
```

- When URLs listing the available versions differ from the download URLs. For example, to limit distfile version checks to the download page for the databases/pgtune port, add:

```
PORTSCOUT= site:http://pgfoundry.org/frs/?group_id=1000416
```

5.8. Dependencies

Many ports depend on other ports. This is a very convenient feature of most Unix-like operating systems, including FreeBSD. Multiple ports can share a common dependency, rather than bundling that dependency with every port or package that needs it. There are seven variables that can be used to ensure that all the required bits will be on the user's machine. There are also some pre-supported dependency variables for common cases, plus a few more to control the behavior of dependencies.

5.8.1. LIB_DEPENDS

This variable specifies the shared libraries this port depends on. It is a list of *lib*:*dir* tuples where *lib* is the name of the shared library, *dir* is the directory in which to find it in case it is not available. For example,

```
LIB_DEPENDS=   libjpeg.so:${PORTSDIR}/graphics/jpeg
```

will check for a shared jpeg library with any version, and descend into the `graphics/jpeg` subdirectory of your ports tree to build and install it if it is not found.

The dependency is checked twice, once from within the `build` target and then from within the `install` target. Also, the name of the dependency is put into the package so that pkg_add(1) will automatically install it if it is not on the user's system.

5.8.2. RUN_DEPENDS

This variable specifies executables or files this port depends on during run-time. It is a list of *path*:*dir*[:target] tuples where *path* is the name of the executable or file, *dir* is the directory in which to find it in case it is not available, and *target* is the target to call in that directory. If *path* starts with a slash (`/`), it is treated as a file and its existence is tested with `test -e`; otherwise, it is assumed to be an executable, and `which -s` is used to determine if the program exists in the search path.

For example,

```
RUN_DEPENDS= ${LOCALBASE}/news/bin/innd:${PORTSDIR}/news/inn \
  xmlcatmgr:${PORTSDIR}/textproc/xmlcatmgr
```

will check if the file or directory `/usr/local/news/bin/innd` exists, and build and install it from the `news/inn` subdirectory of the ports tree if it is not found. It will also see if an executable called `xmlcatmgr` is in the search path, and descend into the `textproc/xmlcatmgr` subdirectory of your ports tree to build and install it if it is not found.

Note

In this case, `innd` is actually an executable; if an executable is in a place that is not expected to be in the search path, you should use the full pathname.

Note

The official search `PATH` used on the ports build cluster is

```
/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin
```

The dependency is checked from within the `install` target. Also, the name of the dependency is put into the package so that pkg_add(1) will automatically install it if it is not on the user's system. The *`target`* part can be omitted if it is the same as `DEPENDS_TARGET`.

A quite common situation is when `RUN_DEPENDS` is literally the same as `BUILD_DEPENDS`, especially if ported software is written in a scripted language or if it requires the same build and run-time environment. In this case, it is both tempting and intuitive to directly assign one to the other:

```
RUN_DEPENDS= ${BUILD_DEPENDS}
```

However, such assignment can pollute run-time dependencies with entries not defined in the port's original `BUILD_DEPENDS`. This happens because of make(1)'s lazy evaluation of variable assignment. Consider a `Makefile` with `USE_*` variables, which are processed by `ports/Mk/bsd.*.mk` to augment initial build dependencies. For example, `USES= gmake` adds devel/gmake to `BUILD_DEPENDS`. To prevent such additional dependencies from polluting `RUN_DEPENDS`, take care to assign with expansion, i.e., expand the value before assigning it to the variable:

```
RUN_DEPENDS:= ${BUILD_DEPENDS}
```

5.8.3. BUILD_DEPENDS

This variable specifies executables or files this port requires to build. Like RUN_DEPENDS , it is a list of *path*:*dir*[:target] tuples. For example,

```
BUILD_DEPENDS= unzip:${PORTSDIR}/archivers/unzip
```

will check for an executable called unzip, and descend into the archivers/unzip subdirectory of your ports tree to build and install it if it is not found.

Note

"build" here means everything from extraction to compilation. The dependency is checked from within the extract target. The *target* part can be omitted if it is the same as DEPENDS_TARGET

5.8.4. FETCH_DEPENDS

This variable specifies executables or files this port requires to fetch. Like the previous two, it is a list of *path*:*dir*[:target] tuples. For example,

```
FETCH_DEPENDS= ncftp2:${PORTSDIR}/net/ncftp2
```

will check for an executable called ncftp2 , and descend into the net/ncftp2 subdirectory of your ports tree to build and install it if it is not found.

The dependency is checked from within the fetch target. The *target* part can be omitted if it is the same as DEPENDS_TARGET .

5.8.5. EXTRACT_DEPENDS

This variable specifies executables or files this port requires for extraction. Like the previous, it is a list of *path*:*dir*[:target] tuples. For example,

```
EXTRACT_DEPENDS= unzip:${PORTSDIR}/archivers/unzip
```

will check for an executable called unzip, and descend into the archivers/unzip subdirectory of your ports tree to build and install it if it is not found.

The dependency is checked from within the extract target. The *target* part can be omitted if it is the same as DEPENDS_TARGET .

Note

Use this variable only if the extraction does not already work (the default assumes `gzip`) and cannot be made to work using `USE_ZIP` or `USE_BZIP2` described in Section 5.8.8, "`USE_*`".

5.8.6. `PATCH_DEPENDS`

This variable specifies executables or files this port requires to patch. Like the previous, it is a list of *`path`*:*`dir`*[:target] tuples. For example,

```
PATCH_DEPENDS= ${NONEXISTENT}:${PORTSDIR}/java/jfc:extract
```

will descend into the `java/jfc` subdirectory of your ports tree to extract it.

The dependency is checked from within the `patch` target. The *`target`* part can be omitted if it is the same as `DEPENDS_TARGET`.

5.8.7. `USES`

There several parameters exist for defining different kind of features and dependencies that the port in question uses. They can be specified by adding the following line to the `Makefile` of the port:

```
USES= feature[:arguments]
```

For the complete list of such values, please see Section 15.1, "Values of `USES`".

Warning

`USES` cannot be assigned after inclusion of `bsd.port.pre.mk`.

5.8.8. `USE_*`

Several variables exist to define common dependencies shared by many ports. Their use is optional, but helps to reduce the verbosity of the port `Makefiles`. Each of them is styled as `USE_*`. These variables may be used only in the port `Makefiles` and `ports/Mk/bsd.*.mk`. They are not meant for user-settable options — use `PORT_OPTIONS` for that purpose.

Note

It is *always* incorrect to set any USE_* in /etc/make.conf . For instance, setting

```
USE_GCC=X.Y
```

(where X.Y is version number) would add a dependency on gccXY for every port, including lang/gccXY itself!

Table 5.2. The USE_* Variables

Variable	Means
USE_BZIP2	The port's tarballs are compressed with bzip2.
USE_ZIP	The port's tarballs are compressed with zip.
USE_GCC	The port requires GCC (gcc or g++) to build. Some ports need any GCC version, some require modern, recent versions. It is typically set to any (in this case, GCC from base would be used on versions of FreeBSD that still have it, or lang/gcc port would be installed when default C/C++ compiler is Clang); or yes (means always use stable, modern GCC from lang/gcc port). The exact version can be also specified, with a value such as 4.7. The minimal required version can be specified as 4.6+. The GCC from the base system is used when it satisfies the requested version, otherwise an appropriate compiler in built from the port, and the CC and CXX variables are adjusted accordingly.

Variables related to gmake and the configure script are described in Section 6.4, "Building Mechanisms", while autoconf, automake and libtool are described in Section 6.5, "Using GNU Autotools". Perl related variables are described in Section 6.7, "Using Perl". X11 variables are listed in Section 6.8, "Using X11". Section 6.9, "Using GNOME" deals with GNOME and Section 6.11, "Using KDE" with KDE related variables. Section 6.12, "Using Java" documents Java variables, while Section 6.13, "Web Applications, Apache and PHP"

contains information on Apache, PHP and PEAR modules. Python is discussed in Section 6.14, “Using Python”, while Ruby in Section 6.17, “Using Ruby”. Section 6.18, “Using SDL” provides variables used for SDL applications and finally, Section 6.22, “Using Xfce” contains information on Xfce.

5.8.9. Minimal Version of a Dependency

A minimal version of a dependency can be specified in any `*_DEPENDS` variable except `LIB_DEPENDS` using the following syntax:

```
p5-Spiffy>=0.26:${PORTSDIR}/devel/p5-Spiffy
```

The first field contains a dependent package name, which must match the entry in the package database, a comparison sign, and a package version. The dependency is satisfied if p5-Spiffy-0.26 or newer is installed on the machine.

5.8.10. Notes on Dependencies

As mentioned above, the default target to call when a dependency is required is `DEPENDS_TARGET`. It defaults to `install`. This is a user variable; it is never defined in a port's `Makefile`. If your port needs a special way to handle a dependency, use the `:target` part of the `*_DEPENDS` variables instead of redefining `DEPENDS_TARGET`.

When you type `make clean`, its dependencies are automatically cleaned too. If you do not wish this to happen, define the variable `NOCLEANDEPENDS` in your environment. This may be particularly desirable if the port has something that takes a long time to rebuild in its dependency list, such as KDE, GNOME or Mozilla.

To depend on another port unconditionally, use the variable `${NONEXISTENT}` as the first field of `BUILD_DEPENDS` or `RUN_DEPENDS`. Use this only when you need to get the source of the other port. You can often save compilation time by specifying the target too. For instance

```
BUILD_DEPENDS= ${NONEXISTENT}:${PORTSDIR}/graphics/jpeg:extract
```

will always descend to the `jpeg` port and extract it.

5.8.11. Circular Dependencies Are Fatal

Important

Do not introduce any circular dependencies into the ports tree!

The ports building technology does not tolerate circular dependencies. If you introduce one, you will have someone, somewhere in the world, whose FreeBSD installation will break almost immediately, with many others quickly to follow. These can really be hard to detect; if in doubt, before you make that change, make sure you have done the following: `cd /usr/ports; make index` . That process can be quite slow on older machines, but you may be able to save a large number of people—including yourself— a lot of grief in the process.

5.8.12. Problems Caused by Automatic Dependencies

Dependencies must be declared either explicitly or by using the OPTIONS framework. Using other methods like automatic detection complicates indexing, which causes problems for port and package management.

Example 5.8. Wrong Declaration of an Optional Dependency

```
.include <bsd.port.pre.mk>

.if exists(${LOCALBASE}/bin/foo)
LIB_DEPENDS= libbar.so:${PORTSDIR}/foo/bar
.endif
```

The problem with trying to automatically add dependencies is that files and settings outside an individual port can change at any time. For example: an index is built, then a batch of ports are installed. But one of the ports installs the tested file. The index is now incorrect, because an installed port unexpectedly has a new dependency. The index may still be wrong even after rebuilding if other ports also determine their need for dependencies based on the existence of other files.

Example 5.9. Correct Declaration of an Optional Dependency

```
OPTIONS_DEFINE= BAR
BAR_DESC= Bar support

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MBAR}
LIB_DEPENDS= libbar.so:${PORTSDIR}/foo/bar
```

```
.endif
```

Testing option variables is the correct method. It will not cause inconsistencies in the index of a batch of ports, provided the options were defined prior to the index build. Simple scripts can then be used to automate the building, installation, and updating of these ports and their packages.

5.8.13. USE_ and WANT_

USE_ variables are set by the port maintainer to define software on which this port depends. A port that needs Firefox would set

```
USE_FIREFOX= yes
```

Some USE_ variables can accept version numbers or other parameters. For example, a port that requires Apache 2.2 would set

```
USE_APACHE= 22
```

For more control over dependencies in some cases, WANT_ variables are available to more precisely specify what is needed. For example, consider the mail/squirrelmail port. This port needs some PHP modules, which are listed in the USE_PHP variable:

```
USE_PHP= session mhash gettext mbstring pcre openssl xml
```

Those modules may be available in CLI or web versions, so the web version is selected with a WANT_ variable:

```
WANT_PHP_WEB= yes
```

Available USE_ and WANT_ variables are defined in the files in /usr/ports/Mk .

5.9. MASTERDIR

If your port needs to build slightly different versions of packages by having a variable (for instance, resolution, or paper size) take different values, create one subdirectory per package to make it easier for users to see what to do, but try to share as many files as possible between ports. Typically you only need a very short Makefile in all but one of the directories if you use variables cleverly. In the sole Makefile, you can use MASTERDIR to specify the directory where the rest of the files are. Also, use a variable as part of PKGNAMESUFFIX so the packages will have different names.

This will be best demonstrated by an example. This is part of japanese/xdvi300/Makefile;

```
PORTNAME= xdvi
```

```
PORTVERSION= 17
PKGNAMEPREFIX= ja-
PKGNAMESUFFIX= ${RESOLUTION}
 -:
# default
RESOLUTION?= 300
.if ${RESOLUTION} -!= 118 && ${RESOLUTION} -!= 240 && \
 ${RESOLUTION} -!= 300 && ${RESOLUTION} -!= 400
 @${ECHO_MSG} -"Error: invalid value for RESOLUTION: ↺
\"${RESOLUTION}\""
 @${ECHO_MSG} -"Possible values are: 118, 240, 300 (default) and ↺
400."
 @${FALSE}
.endif
```

japanese/xdvi300 also has all the regular patches, package files, etc. If you type `make` there, it will take the default value for the resolution (300) and build the port normally.

As for other resolutions, this is the *entire* `xdvi118/Makefile`:

```
RESOLUTION= 118
MASTERDIR= ${.CURDIR}/../xdvi300

.include -"${MASTERDIR}/Makefile"
```

(`xdvi240/Makefile` and `xdvi400/Makefile` are similar). The `MASTERDIR` definition tells `bsd.port.mk` that the regular set of subdirectories like `FILESDIR` and `SCRIPTDIR` are to be found under `xdvi300`. The `RESOLUTION=118` line will override the `RESOLUTION=300` line in `xdvi300/Makefile` and the port will be built with resolution set to 118.

5.10. Man Pages

If your port anchors its man tree somewhere other than `PREFIX`, you can use `MANDIRS` to specify those directories. Note that the files corresponding to manual pages should be placed in `pkg-plist` along with the rest of the files. The purpose of `MANDIRS` is to enable automatic compression of manual pages, therefore the file names should be suffixed with `.gz`.

5.11. Info Files

If your package needs to install GNU info files, they should be listed in the `INFO` variable (without the trailing `.info`), one entry per document. These files are assumed to be installed to `PREFIX/INFO_PATH`. You can change `INFO_PATH` if your package uses a different location. However, this is not recommended. These entries contain just the path relative to `PREFIX/INFO_PATH`. For example, lang/gcc34 installs info files to `PREFIX/INFO_PATH/gcc34`, and `INFO` will be something like this:

```
INFO= gcc34/cpp gcc34/cppinternals gcc34/g77 -...
```

Appropriate installation/de-installation code will be automatically added to the temporary `pkg-plist` before package registration.

5.12. Makefile Options

Many applications can be built with optional or differing configurations. Examples include choice of natural (human) language, GUI versus command-line, or type of database to support. Users may need a different configuration than the default, so the ports system provides hooks the port author can use to control which variant will be built. Supporting these options properly will make users happy, and effectively provide two or more ports for the price of one.

5.12.1. Knobs

5.12.1.1. `WITH_*` and `WITHOUT_*`

These variables are designed to be set by the system administrator. There are many that are standardized in the `ports/KNOBS` file.

When creating a port, do not make knob names specific to a given application. For example in Avahi port, use `WITHOUT_MDNS` instead of `WITHOUT_AVAHI_MDNS` .

Note

You should not assume that a `WITH_*` necessarily has a corresponding `WITHOUT_*` variable and vice versa. In general, the default is simply assumed.

Note

Unless otherwise specified, these variables are only tested for being set or not set, rather than being set to a specific value such as `YES` or `NO`.

Table 5.3. Common WITH_* and WITHOUT_* Variables

Variable	Means
WITH_OPENSSL_BASE	Use the version of OpenSSL in the base system.
WITH_OPENSSL_PORT	Installs the version of OpenSSL from security/openssl, even if the base is up to date.

5.12.1.2. Knob Naming

Porters should use like-named knobs, both for the benefit of end-users and to help keep the number of knob names down. A list of popular knob names can be found in the KNOBS file.

Knob names should reflect what the knob is and does. When a port has a lib-prefix in the PORTNAME the lib-prefix should be dropped in knob naming.

5.12.2. OPTIONS

5.12.2.1. Background

The OPTIONS_* variables give the user installing the port a dialog showing the available options, and then saves those options to /var/db/ports/${UNIQUENAME}/options . The next time the port is built, the options are reused.

When the user runs make config (or runs make build for the first time), the framework checks for /var/db/ports/${UNIQUENAME}/options . If that file does not exist, the values of OPTIONS_* are used, and a dialog box is displayed where the options can be enabled or disabled. Then the options file is saved and the configured variables are used when building the port.

If a new version of the port adds new OPTIONS, the dialog will be presented to the user with the saved values of old OPTIONS prefilled.

make showconfig shows the saved configuration. Use make rmconfig to remove the saved configuration.

5.12.2.2. Syntax

OPTIONS_DEFINE contains a list of OPTIONS to be used. These are independent of each other and are not grouped:

```
OPTIONS_DEFINE= OPT1 OPT2
```

Once defined, OPTIONS are described (optional, but strongly recommended):

```
OPT1_DESC= Describe OPT1
OPT2_DESC= Describe OPT2
```

```
OPT3_DESC= Describe OPT3
OPT4_DESC= Describe OPT4
OPT5_DESC= Describe OPT5
OPT6_DESC= Describe OPT6
```

Tip

`ports/Mk/bsd.options.desc.mk` has descriptions for many common `OPTIONS`; there is usually no need to override these.

Tip

When describing options, view it from the perspective of the user: "What does it do?" and "Why would I want to enable this?" Do not just repeat the name. For example, describing the `NLS` option as "include NLS support" does not help the user, who can already see the option name but may not know what it means. Describing it as "Native Language Support via gettext utilities" is much more helpful.

`OPTIONS` can be grouped as radio choices, where only one choice from each group is allowed:

```
OPTIONS_SINGLE=  SG1
OPTIONS_SINGLE_SG1= OPT3 OPT4
```

`OPTIONS` can be grouped as radio choices, where none or only one choice from each group is allowed:

```
OPTIONS_RADIO=  RG1
OPTIONS_RADIO_RG1= OPT7 OPT8
```

`OPTIONS` can also be grouped as "multiple-choice" lists, where *at least one* option must be enabled:

```
OPTIONS_MULTI=  MG1
OPTIONS_MULTI_MG1= OPT5 OPT6
```

`OPTIONS` can also be grouped as "multiple-choice" lists, where none or any option can be enabled:

```
OPTIONS_GROUP=  GG1
OPTIONS_GROUP_GG1= OPT9 OPT10
```

`OPTIONS` are unset by default, unless they are listed in `OPTIONS_DEFAULT` :

```
OPTIONS_DEFAULT= OPT1 OPT3 OPT6
```

`OPTIONS` definitions must appear before the inclusion of `bsd.port.options.mk` . The `PORT_OPTIONS` variable can only be tested after the inclusion of `bsd.port.options.mk` . Inclusion of `bsd.port.pre.mk` can be used instead, too, and is still widely used in ports written before the introduction of `bsd.port.options.mk` . But be aware that some variables will not work as expected after the inclusion of `bsd.port.pre.mk` , typically some `USE_*` flags.

Example 5.10. Simple Use of **OPTIONS**

```
OPTIONS_DEFINE= FOO BAR
FOO_DESC= Enable option foo
BAR_DESC= Support feature bar

OPTIONS_DEFAULT=FOO

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MFOO}
CONFIGURE_ARGS+=--with-foo
.else
CONFIGURE_ARGS+=--without-foo
.endif

.if ${PORT_OPTIONS:MBAR}
RUN_DEPENDS+= bar:${PORTSDIR}/bar/bar
.endif

.include <bsd.port.mk>
```

Example 5.11. Check for Unset Port **OPTIONS**

```
.if -! ${PORT_OPTIONS:MEXAMPLES}
CONFIGURE_ARGS+=--without-examples
.endif
```

Example 5.12. Practical Use of **OPTIONS**

```
OPTIONS_DEFINE=  EXAMPLES
```

```
OPTIONS_SINGLE=  BACKEND
OPTIONS_SINGLE_BACKEND= MYSQL PGSQL BDB

OPTIONS_MULTI=  AUTH
OPTIONS_MULTI_AUTH= LDAP PAM SSL

EXAMPLES_DESC=  Install extra examples
MYSQL_DESC=  Use MySQL as backend
PGSQL_DESC=  Use PostgreSQL as backend
BDB_DESC=  Use Berkeley DB as backend
LDAP_DESC=  Build with LDAP authentication support
PAM_DESC=  Build with PAM support
SSL_DESC=  Build with OpenSSL support

OPTIONS_DEFAULT= PGSQL LDAP SSL

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MPGSQL}
USE_PGSQL=  yes
CONFIGURE_ARGS+= --with-postgres
.else
CONFIGURE_ARGS+= --without-postgres
.endif

.if ${PORT_OPTIONS:MICU}
LIB_DEPENDS+= libicuuc.so:${PORTSDIR}/devel/icu
.endif

.if -! ${PORT_OPTIONS:MEXAMPLES}
CONFIGURE_ARGS+= --without-examples
.endif

# Check other OPTIONS

.include <bsd.port.mk>
```

5.12.2.3. Default Options

The following options are always on by default.

- `DOCS` — build and install documentation.
- `NLS` — Native Language Support.
- `EXAMPLES` — build and install examples.
- `IPV6` — IPv6 protocol support.

Note

There is no need to add these to `OPTIONS_DEFAULT`. To have them show up in the options selection dialog, however, they must be added to `OPTIONS_DEFINE`.

5.12.3. Feature Auto-Activation

When using a GNU configure script, keep an eye on which optional features are activated by auto-detection. Explicitly disable optional features you do not wish to be used by passing respective `--without-xxx` or `--disable-xxx` in `CONFIGURE_ARGS`.

Example 5.13. Wrong Handling of an Option

```
.if ${PORT_OPTIONS:MFOO}
LIB_DEPENDS+=  libfoo.so:${PORTSDIR}/devel/foo
CONFIGURE_ARGS+= --enable-foo
.endif
```

In the example above, imagine a library libfoo is installed on the system. The user does not want this application to use libfoo, so he toggled the option off in the `make config` dialog. But the application's configure script detects the library present in the system and includes its support in the resulting executable. Now when the user decides to remove libfoo from the system, the ports system does not protest (no dependency on libfoo was recorded) but the application breaks.

Example 5.14. Correct Handling of an Option

```
.if ${PORT_OPTIONS:MFOO}
LIB_DEPENDS+=  libfoo.so:${PORTSDIR}/devel/foo
CONFIGURE_ARGS+= --enable-foo
.else
CONFIGURE_ARGS+= --disable-foo
.endif
```

In the second example, the library libfoo is explicitly disabled. The configure script does not enable related features in the application, despite library's presence in the system.

Note

Under some circumstances, the shorthand conditional syntax can cause problems with complex constructs. If you receive errors such as `Malformed conditional`, an alternative syntax can be used.

```
.if -!empty(VARIABLE:MVALUE)
# as an alternative to
.if ${VARIABLE:MVALUE}
```

5.12.4. Options Helpers

There are some macros to help simplify conditional values which differ based on the options set.

If `OPTIONS_SUB` is set to `yes` then each of the options added to `OPTIONS_DEFINE` will be added to `PLIST_SUB` , for example:

```
OPTIONS_DEFINE= OPT1
OPTIONS_SUB= yes
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MOPT1}
PLIST_SUB+= OPT1=""
.else
PLIST_SUB+= OPT1="@comment -"
.endif
```

If `X_CONFIGURE_ENABLE` is set then `--enable-${X_CONFIGURE_ENABLE}` or `--disable-${X_CONFIGURE_ENABLE}` will be added to `CONFIGURE_ARGS` depending on the value of the option`X`, for example:

```
OPTIONS_DEFINE= OPT1
OPT1_CONFIGURE_ENABLE= test
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1

.include <bsd.port.options.mk>
```

```
.if ${PORT_OPTIONS:MOPT1}
CONFIGURE_ARGS+= --enable-test
.else
CONFIGURE_ARGS+= --disable-test
.endif
```

If `X_CONFIGURE_WITH` is set then `--with-${X_CONFIGURE_WITH}` or `--without-${X_CONFIGURE_WITH}` will be added to `CONFIGURE_ARGS` depending on the status of the option `X`, for example:

```
OPTIONS_DEFINE= OPT1
OPT1_CONFIGURE_WITH= test
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MOPT1}
CONFIGURE_ARGS+= --with-test
.else
CONFIGURE_ARGS+= --without-test
.endif
```

If `X_CONFIGURE_ON` is set then its value will be appended to `CONFIGURE_ARGS` depending on the status of the option `X`, for example:

```
OPTIONS_DEFINE= OPT1
OPT1_CONFIGURE_ON= --add-test
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MOPT1}
CONFIGURE_ARGS+= --add-test
.endif
```

If `X_CONFIGURE_OFF` is set then its value will be appended to `CONFIGURE_ARGS` depending on the status of the option `X`, for example:

```
OPTIONS_DEFINE= OPT1
OPT1_CONFIGURE_OFF= --no-test
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1
.include <bsd.port.options.mk>
.if -! ${PORT_OPTIONS:MOPT1}
```

```
CONFIGURE_ARGS+= --no-test
.endif
```

If `X_CMAKE_ON` is set then its value will be appended to `CMAKE_ARGS` depending on the status of the option `X`, for example:

```
OPTIONS_DEFINE= OPT1
OPT1_CMAKE_ON= -DTEST:BOOL=true
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MOPT1}
CMAKE_ARGS+= -DTEST:BOOL=true
.endif
```

If `X_CMAKE_OFF` is set then its value will be appended to `CMAKE_ARGS` depending on the status of the option `X`, for example:

```
OPTIONS_DEFINE= OPT1
OPT1_CMAKE_OFF= -DTEST:BOOL=false
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1

.include <bsd.port.options.mk>

.if -! ${PORT_OPTIONS:MOPT1}
CMAKE_ARGS+= -DTEST:BOOL=false
.endif
```

For any of the following variables:

- `ALL_TARGET`
- `CATEGORIES`
- `CFLAGS`
- `CPPFLAGS`
- `CXXFLAGS`
- `CONFIGURE_ENV`
- `DISTFILES`
- `EXTRA_PATCHES`

- INSTALL_TARGET
- LDFLAGS
- MAKE_ARGS
- MAKE_ENV
- PATCH_SITES
- PATCHFILES
- PLIST_FILES
- PLIST_DIRS
- PLIST_DIRSTRY
- USES

If X_ABOVEVARIABLE is defined then its value will be appended to ABOVEVARIABLE depending on the status of the option X, for example:

```
OPTIONS_DEFINE= OPT1
OPT1_USES= gmake
OPT1_CFLAGS= -DTEST
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MOPT1}
USES+=  gmake
CFLAGS+= -DTEST
.endif
```

For any of the following dependency type:

- PKG_DEPENDS
- EXTRACT_DEPENDS
- PATCH_DEPENDS
- FETCH_DEPENDS
- BUILD_DEPENDS
- LIB_DEPENDS

- RUN_DEPENDS

If X_ABOVEVARIABLE is defined then its value will be appended to ABOVEVARIABLE depending on the status of the option X, for example:

```
OPTIONS_DEFINE= OPT1
OPT1_LIB_DEPENDS= liba.so:${PORTSDIR}/devel/a
```

is equivalent to:

```
OPTIONS_DEFINE= OPT1

.include <bsd.port.options.mk>

.if ${PORT_OPTIONS:MOPT1}
LIB_DEPENDS+= liba.so:${PORTSDIR}/devel/a
.endif
```

5.13. Specifying the Working Directory

Each port is extracted in to a working directory, which must be writable. The ports system defaults to having the DISTFILES unpack in to a directory called ${DISTNAME}. In other words, if you have set:

```
PORTNAME= foo
PORTVERSION= 1.0
```

then the port's distribution files contain a top-level directory, foo-1.0, and the rest of the files are located under that directory.

There are a number of variables you can override if that is not the case.

5.13.1. WRKSRC

The variable lists the name of the directory that is created when the application's distfiles are extracted. If our previous example extracted into a directory called foo (and not foo-1.0) you would write:

```
WRKSRC= ${WRKDIR}/foo
```

or possibly

```
WRKSRC= ${WRKDIR}/${PORTNAME}
```

5.13.2. NO_WRKSUBDIR

If the port does not extract in to a subdirectory at all then you should set NO_WRKSUBDIR to indicate that.

```
NO_WRKSUBDIR= yes
```

5.14. Conflict Handling

There are three different variables to register a conflict between packages and ports: CONFLICTS, CONFLICTS_INSTALL and CONFLICTS_BUILD .

Note

The conflict variables automatically set the variable IGNORE, which is more fully documented in Section 12.13, “Marking a Port Not Installable with BROKEN, FORBIDDEN, or IGNORE”.

When removing one of several conflicting ports, it is advisable to retain the CONFLICTS entries in those other ports for a few months to cater for users who only update once in a while.

5.14.1. CONFLICTS_INSTALL

If your package cannot coexist with other packages (because of file conflicts, runtime incompatibilities, etc.), list the other package names in the CONFLICTS_INSTALL variable. You can use shell globs like * and ? here. Package names should be enumerated the same way they appear in /var/db/pkg . Please make sure that CONFLICTS_INSTALL does not match this port's package itself. Otherwise enforcing its installation with FORCE_PKG_REGISTER will no longer work. The CONFLICTS_INSTALL check is done after the build stage and prior to the install stage.

5.14.2. CONFLICTS_BUILD

If your port cannot be built if a certain port is already installed, list the other port names in the CONFLICTS_BUILD variable. You can use shell globs like * and ? here. Package names should be enumerated the same way they appear in /var/db/pkg . The CONFLICTS_BUILD check is done prior to the build stage. Build conflicts are not recorded in the resulting package.

5.14.3. CONFLICTS

If your port cannot be built if a certain port is already installed and the resulting package cannot coexist with the other package, list the other package name in the CONFLICTS variable. You can use shell globs like * and ? here. Packages names should be enumerated the same way they appear in /var/db/pkg . Please make sure that CONFLICTS_INSTALL does not match this port's package itself. Otherwise enforcing its installation with

FORCE_PKG_REGISTER will no longer work. The CONFLICTS check is done prior to the build stage and prior to the install stage.

5.15. Installing Files

5.15.1. INSTALL_* Macros

Use the macros provided in bsd.port.mk to ensure correct modes of files in the port's *-install targets. Set ownership directly in pkg-plist with the corresponding entries, such as @owner owner and @group group. These operators work until being overridden, or until the end of pkg-plist, so do not forget to reset them after they are no longer needed. The default ownership is root:wheel.

- INSTALL_PROGRAM is a command to install binary executables.
- INSTALL_SCRIPT is a command to install executable scripts.
- INSTALL_LIB is a command to install shared libraries.
- INSTALL_KLD is a command to install kernel loadable modules. Some architectures do not like having the modules stripped, so use this command instead of INSTALL_PROGRAM.
- INSTALL_DATA is a command to install sharable data.
- INSTALL_MAN is a command to install manpages and other documentation (it does not compress anything).

These are basically the install command with all the appropriate flags.

5.15.2. Stripping Binaries and Shared Libraries

Do not strip binaries manually unless you have to. All binaries should be stripped, but the INSTALL_PROGRAM macro will install and strip a binary at the same time (see the next section). The INSTALL_LIB macro does the same thing to shared libraries.

If you need to strip a file, but wish to use neither INSTALL_PROGRAM nor INSTALL_LIB macros, ${STRIP_CMD} will strip your program or shared library. This is typically done within the post-install target. For example:

```
post-install:
   ${STRIP_CMD} ${STAGEDIR}${PREFIX}/bin/xdl
```

Use the file(1) command on the installed executable to check whether the binary is stripped or not. If it does not say not stripped, it is stripped. Additionally, strip(1) will not strip a previously stripped program; it will instead exit cleanly.

5.15.3. Installing a Whole Tree of Files

Sometimes, a large number of files must be installed while preserving their hierarchical organization. For example, copying over a whole directory tree from `WRKSRC` to a target directory under `PREFIX`. Note that `PREFIX`, `EXAMPLESDIR`, `DATADIR`, and other path varialbes must always be prepended with `STAGEDIR` to respect staging (see Section 6.1, "Staging").

Two macros exist for this situation. The advantage of using these macros instead of `cp` is that they guarantee proper file ownership and permissions on target files. The first macro, `COPYTREE_BIN`, will set all the installed files to be executable, thus being suitable for installing into `PREFIX/bin`. The second macro, `COPYTREE_SHARE`, does not set executable permissions on files, and is therefore suitable for installing files under `PREFIX/share` target.

```
post-install:
   ${MKDIR} ${STAGEDIR}${EXAMPLESDIR}
   (cd ${WRKSRC}/examples && ${COPYTREE_SHARE} -. ↺
${STAGEDIR}${EXAMPLESDIR})
```

This example will install the contents of `examples` directory in the vendor distfile to the proper examples location of your port.

```
post-install:
   ${MKDIR} ${STAGEDIR}${DATADIR}/summer
   (cd ${WRKSRC}/temperatures && ${COPYTREE_SHARE} -"June July ↺
August" ${STAGEDIR}${DATADIR}/summer)
```

And this example will install the data of summer months to the `summer` subdirectory of a `DATADIR`.

Additional `find` arguments can be passed via the third argument to the `COPYTREE_*` macros. For example, to install all files from the first example except Makefiles, one can use the following command.

```
post-install:
   ${MKDIR} ${STAGEDIR}${EXAMPLESDIR}
  (cd ${WRKSRC}/examples && \
  ${COPYTREE_SHARE} -. ${STAGEDIR}${EXAMPLESDIR} -"! --name Makefile")
```

These macros do not add the installed files to `pkg-plist`. They must be added manually. For optional documentation (`PORTDOCS`, see Section 5.15.4, "Install Additional Documentation") and examples (`PORTEXAMPLES`), the `%%PORTDOCS%%` or `%%PORTEXAMPLES%%` prefixes must be prepended in `pkg-plist`.

5.15.4. Install Additional Documentation

If your software has some documentation other than the standard man and info pages that you think is useful for the user, install it under `PREFIX/share/doc`. This can be done, like the previous item, in the `post-install` target.

Create a new directory for your port. The directory name should reflect what the port is. This usually means PORTNAME. However, if you think the user might want different versions of the port to be installed at the same time, you can use the whole PKGNAME.

Since only the files listed in pkg-plist are installed, it is safe to always install documentation to STAGEDIR (see Section 6.1, "Staging"). Hence .if blocks are only needed when the installed files are large enough to cause significant I/O overhead.

```
post-install:
   ${MKDIR} ${STAGEDIR}${DOCSDIR}
   ${INSTALL_MAN} ${WRKSRC}/docs/xvdocs.ps ${STAGEDIR}${DOCSDIR}
```

Here are some handy variables and how they are expanded by default when used in the Makefile:

- DATADIR gets expanded to PREFIX/share/PORTNAME .
- DATADIR_REL gets expanded to share/PORTNAME .
- DOCSDIR gets expanded to PREFIX/share/doc/PORTNAME .
- DOCSDIR_REL gets expanded to share/doc/PORTNAME .
- EXAMPLESDIR gets expanded to PREFIX/share/examples/PORTNAME .
- EXAMPLESDIR_REL gets expanded to share/examples/PORTNAME .

Note

The DOCS option only controls additional documentation installed in DOCSDIR . It does not apply to standard man pages and info pages. Things installed in DATADIR and EXAMPLESDIR are controlled by DATA and EXAMPLES options, respectively.

These variables are exported to PLIST_SUB . Their values will appear there as pathnames relative to PREFIX if possible. That is, share/doc/PORTNAME will be substituted for %%DOCSDIR%% in the packing list by default, and so on. (See more on pkg-plist substitution here.)

All conditionally installed documentation files and directories should be included in pkg-plist with the %%PORTDOCS%% prefix, for example:

```
%%PORTDOCS%%%%DOCSDIR%%/AUTHORS
%%PORTDOCS%%%%DOCSDIR%%/CONTACT
%%PORTDOCS%%@dirrm %%DOCSDIR%%
```

As an alternative to enumerating the documentation files in pkg-plist , a port can set the variable PORTDOCS to a list of file names and shell glob patterns to add to the final packing list. The names will be relative to DOCSDIR . Therefore, a port that utilizes PORTDOCS and uses a non-default location for its documentation should set DOCSDIR accordingly. If a directory is listed in PORTDOCS or matched by a glob pattern from this variable, the entire subtree of contained files and directories will be registered in the final packing list. If the DOCS option has been unset then files and directories listed in PORTDOCS would not be installed or added to port packing list. Installing the documentation at PORTDOCS as shown above remains up to the port itself. A typical example of utilizing PORTDOCS looks as follows:

```
PORTDOCS= README.* ChangeLog docs/*
```

Note

The equivalents of PORTDOCS for files installed under DATADIR and EXAMPLESDIR are PORTDATA and PORTEXAMPLES, respectively.

You can also use the pkg-message file to display messages upon installation. See the section on using pkg-message for details. The pkg-message file does not need to be added to pkg-plist .

5.15.5. Subdirectories Under PREFIX

Try to let the port put things in the right subdirectories of PREFIX. Some ports lump everything and put it in the subdirectory with the port's name, which is incorrect. Also, many ports put everything except binaries, header files and manual pages in a subdirectory of lib, which does not work well with the BSD paradigm. Many of the files should be moved to one of the following: etc (setup/configuration files), libexec (executables started internally), sbin (executables for superusers/managers), info (documentation for info browser) or share (architecture independent files). See hier(7) for details; the rules governing /usr pretty much apply to /usr/local too. The exception are ports dealing with USENET “news”. They may use PREFIX/news as a destination for their files.

Chapter 6. Special Considerations

There are some more things you have to take into account when you create a port. This section explains the most common of those.

6.1. Staging

`bsd.port.mk` expects ports to work with a "stage directory". This means that a port should not install files directly to the regular destination directories (that is, under `PREFIX`, for example) but instead into a separate directory from which the package is then built. In many cases, this does not require root privileges, making it possible to build packages as an unprivileged user. With staging, the port is built and installed into the stage directory, `STAGEDIR`. A package is created from the stage directory and then installed on the system. Automake tools refer to this concept as `DESTDIR`, but in FreeBSD, `DESTDIR` has a different meaning (see Section 9.4, "`PREFIX` and `DESTDIR`").

When a port still requires system-wide privileges in order to run the `package` target, this line must be added to the `Makefile`:

```
NEED_ROOT= yes
```

Meta ports, or ports that do not install files themselves but only depend on other ports, should avoid needlessly extracting the mtree(8) to the stage directory. This is the basic directory layout of the package, and these empty directories will be seens as orphans. To prevent mtree(8) extraction, add this line:

```
NO_MTREE= yes
```

Staging is enabled by prepending the `STAGEDIR` variable to paths used in the `pre-install`, `do-install`, and `post-install` targets (see the examples through the book). Typically, this includes `PREFIX`, `ETCDIR`, `DATADIR`, `EXAMPLESDIR`, `MANPREFIX`, `DOCSDIR`, and so on. Directories should be created as part of the `post-install` target. Avoid using absolute paths whenever possible.

When creating a symlink, `STAGEDIR` should be prepended to the target path only. For example:

```
${LN} --sf libfoo.so.42 ${STAGEDIR}${PREFIX}/lib/libfoo.so
```

The source path `${PREFIX}/lib/libfoo.so.42` looks fine but could, in fact, be incorrect. Absolute paths can point to a wrong location, like when a remote file system has been

mounted with NFS under a non-root mount point. Relative paths are less fragile, and often much shorter.

Ports that install kernel modules must prepend the `STAGEDIR` variable to their destination, by default `/boot/modules` .

6.2. Shared Libraries

If your port installs one or more shared libraries, define a `USE_LDCONFIG` make variable, which will instruct a `bsd.port.mk` to run `${LDCONFIG} -m` on the directory where the new library is installed (usually `PREFIX/lib`) during `post-install` target to register it into the shared library cache. This variable, when defined, will also facilitate addition of an appropriate `@exec /sbin/ldconfig -m` and `@unexec /sbin/ldconfig -R` pair into your `pkg-plist` file, so that a user who installed the package can start using the shared library immediately and de-installation will not cause the system to still believe the library is there.

```
USE_LDCONFIG= yes
```

If you need, you can override the default directory by setting the `USE_LDCONFIG` value to a list of directories into which shared libraries are to be installed. For example if your port installs shared libraries into `PREFIX/lib/foo` and `PREFIX/lib/bar` directories you could use the following in your `Makefile`:

```
USE_LDCONFIG= ${PREFIX}/lib/foo ${PREFIX}/lib/bar
```

Please double-check, often this is not necessary at all or can be avoided through `-rpath` or setting `LD_RUN_PATH` during linking (see lang/moscow_ml for an example), or through a shell-wrapper which sets `LD_LIBRARY_PATH` before invoking the binary, like www/seamonkey does.

When installing 32-bit libraries on 64-bit system, use `USE_LDCONFIG32` instead.

Try to keep shared library version numbers in the `libfoo.so.0` format. Our runtime linker only cares for the major (first) number.

When the major library version number increments in the update to the new port version, all other ports that link to the affected library should have their `PORTREVISION` incremented, to force recompilation with the new library version.

6.3. Ports with Distribution Restrictions

Licenses vary, and some of them place restrictions on how the application can be packaged, whether it can be sold for profit, and so on.

Important

It is your responsibility as a porter to read the licensing terms of the software and make sure that the FreeBSD project will not be held accountable for violating them by redistributing the source or compiled binaries either via FTP/HTTP or CD-ROM. If in doubt, please contact the FreeBSD ports mailing list.

In situations like this, the variables described in the following sections can be set.

6.3.1. NO_PACKAGE

This variable indicates that we may not generate a binary package of the application. For instance, the license may disallow binary redistribution, or it may prohibit distribution of packages created from patched sources.

However, the port's DISTFILES may be freely mirrored on FTP/HTTP. They may also be distributed on a CD-ROM (or similar media) unless NO_CDROM is set as well.

NO_PACKAGE should also be used if the binary package is not generally useful, and the application should always be compiled from the source code. For example, if the application has configuration information that is site specific hard coded in to it at compile time, set NO_PACKAGE .

NO_PACKAGE should be set to a string describing the reason why the package should not be generated.

6.3.2. NO_CDROM

This variable alone indicates that, although we are allowed to generate binary packages, we may put neither those packages nor the port's DISTFILES onto a CD-ROM (or similar media) for resale. However, the binary packages and the port's DISTFILES will still be available via FTP/HTTP.

If this variable is set along with NO_PACKAGE , then only the port's DISTFILES will be available, and only via FTP/HTTP.

NO_CDROM should be set to a string describing the reason why the port cannot be redistributed on CD-ROM. For instance, this should be used if the port's license is for “non-commercial” use only.

6.3.3. NOFETCHFILES

Files defined in the NOFETCHFILES variable are not fetchable from any of the MASTER_SITES . An example of such a file is when the file is supplied on CD-ROM by the vendor.

Tools which check for the availability of these files on the MASTER_SITES should ignore these files and not report about them.

6.3.4. RESTRICTED

Set this variable alone if the application's license permits neither mirroring the application's DISTFILES nor distributing the binary package in any way.

NO_CDROM or NO_PACKAGE should not be set along with RESTRICTED since the latter variable implies the former ones.

RESTRICTED should be set to a string describing the reason why the port cannot be redistributed. Typically, this indicates that the port contains proprietary software and that the user will need to manually download the DISTFILES, possibly after registering for the software or agreeing to accept the terms of an EULA.

6.3.5. RESTRICTED_FILES

When RESTRICTED or NO_CDROM is set, this variable defaults to ${DISTFILES} ${PATCHFILES}, otherwise it is empty. If only some of the distribution files are restricted, then set this variable to list them.

Note that the port committer should add an entry to /usr/ports/LEGAL for every listed distribution file, describing exactly what the restriction entails.

6.3.6. Examples

The preferred way to state "the distfiles for this port must be fetched manually" is as follows:

```
.if -!exists(${DISTDIR}/${DISTNAME}${EXTRACT_SUFX})
IGNORE= may not be redistributed because of licensing reasons. ↺
Please visit some-website to accept their license and download ↺
${DISTFILES} into ${DISTDIR}
.endif
```

This both informs the user, and sets the proper metadata on the user's machine for use by automated programs.

Note that this stanza must be preceded by an inclusion of bsd.port.pre.mk .

6.4. Building Mechanisms

6.4.1. Building Ports in Parallel

The FreeBSD ports framework supports parallel building using multiple `make` sub-processes, which allows SMP systems to utilize all of their available CPU power, allowing port builds to be faster and more effective.

This is achieved by passing `-jX` flag to make(1) running on vendor code. Unfortunately, not all ports handle parallel building well. Therefore it is required to explicitly enable this feature by adding `MAKE_JOBS_SAFE=yes` somewhere below the dependency declaration section of the `Makefile`.

Another option for controlling this feature from the maintainer's point of view is the `MAKE_JOBS_UNSAFE=yes` variable. It is used when a port is known to be broken with `-jX` and a user forces the use of multi processor compilations for all ports in `/etc/make.conf` with the `FORCE_MAKE_JOBS=yes` variable.

6.4.2. `make`, `gmake`, and `imake`

If your port uses GNU make, set `USES= gmake`.

Table 6.1. Variables for Ports Related to gmake

Variable	Means
`USES= gmake`	The port requires `gmake` to build.
`GMAKE`	The full path for `gmake` if it is not in the `PATH`.

If your port is an X application that creates `Makefile` files from `Imakefile` files using imake, then set `USES= imake`. This will cause the configure stage to automatically do an `xmkmf -a`. If the `-a` flag is a problem for your port, set `XMKMF=xmkmf`. If the port uses imake but does not understand the `install.man` target, `NO_INSTALL_MANPAGES=yes` should be set.

If your port's source `Makefile` has something else than `all` as the main build target, set `ALL_TARGET` accordingly. Same goes for `install` and `INSTALL_TARGET`.

6.4.3. `configure` Script

If your port uses the `configure` script to generate `Makefile` files from `Makefile.in` files, set `GNU_CONFIGURE=yes`. If you want to give extra arguments to the `configure` script (the default argument is `--prefix=${PREFIX} --infodir=${PREFIX}/${INFO_PATH} --mandir=${MANPREFIX}/man --build=${CONFIGURE_TARGET}`), set those extra arguments in `CONFIGURE_ARGS`. Extra environment variables can be passed using `CONFIGURE_ENV` variable.

Table 6.2. Variables for Ports That Use configure

Variable	Means
GNU_CONFIGURE	The port uses configure script to prepare build.
HAS_CONFIGURE	Same as GNU_CONFIGURE , except default configure target is not added to CONFIGURE_ARGS .
CONFIGURE_ARGS	Additional arguments passed to configure script.
CONFIGURE_ENV	Additional environment variables to be set for configure script run.
CONFIGURE_TARGET	Override default configure target. Default value is ${MACHINE_ARCH}-portbld-freebsd${OSREL} .

6.4.4. Using cmake

For ports that use CMake, define USES= cmake , or USES= cmake:outsource to build in a separate directory (see below).

Table 6.3. Variables for Ports That Use cmake

Variable	Means
CMAKE_ARGS	Port specific CMake flags to be passed to the cmake binary.
CMAKE_BUILD_TYPE	Type of build (CMake predefined build profiles). Default is Release, or Debug if WITH_DEBUG is set.
CMAKE_ENV	Environment variables to be set for the cmake binary. Default is ${CONFIGURE_ENV} .
CMAKE_SOURCE_PATH	Path to the source directory. Default is ${WRKSRC} .

CMake supports the following build profiles: Debug, Release, RelWithDebInfo and MinSizeRel. Debug and Release profiles respect system *FLAGS , RelWithDebInfo and MinSizeRel will set CFLAGS to -O2 -g and -Os -DNDEBUG correspondingly. The lower-cased value of CMAKE_BUILD_TYPE is exported to the PLIST_SUB and should be used if port installs *.cmake files depending on the build type (see deskutils/strigi for an example). Please note that some projects may define their own build profiles and/or force particular build type by setting CMAKE_BUILD_TYPE in CMakeLists.txt files. In order to

make a port for such a project respect `CFLAGS` and `WITH_DEBUG`, the `CMAKE_BUILD_TYPE` definitions must be removed from those files.

Most CMake-based projects support an out-of-source method of building. The out-of-source build for a port can be requested by using the `:outsource` suffix. When enabled, `CONFIGURE_WRKSRC`, `BUILD_WRKSRC` and `INSTALL_WRKSRC` will be set to `${WRKDIR}/.build` and this directory will be used to keep all files generated during configuration and build stages, leaving the source directory intact.

Example 6.1. `USES= cmake` Example

The following snippet demonstrates the use of CMake for a port. `CMAKE_SOURCE_PATH` is not usually required, but can be set when the sources are not located in the top directory, or if only a subset of the project is intended to be built by the port.

```
USES=   cmake:outsource
CMAKE_SOURCE_PATH= ${WRKSRC}/subproject
```

6.4.5. Using `scons`

If your port uses SCons, define `USE_SCONS=yes`.

Table 6.4. Variables for Ports That Use `scons`

Variable	Means
`SCONS_ARGS`	Port specific SCons flags passed to the SCons environment.
`SCONS_BUILDENV`	Variables to be set in system environment.
`SCONS_ENV`	Variables to be set in SCons environment.
`SCONS_TARGET`	Last argument passed to SCons, similar to `MAKE_TARGET`.

To make third party `SConstruct` respect everything that is passed to SCons in `SCONS_ENV` (that is, most importantly, `CC/CXX/CFLAGS/CXXFLAGS`), patch the `SConstruct` so build `Environment` is constructed like this:

```
env = Environment(**ARGUMENTS)
```

It may be then modified with `env.Append` and `env.Replace`.

6.5. Using GNU Autotools

6.5.1. Introduction

The various GNU autotools provide an abstraction mechanism for building a piece of software over a wide variety of operating systems and machine architectures. Within the Ports Collection, an individual port can make use of these tools via a simple construct:

```
USE_AUTOTOOLS= tool:version [:operation ] -...
```

At the time of writing, *tool* can be one of `libtool` , `libltdl`, `autoconf`, `autoheader`, `automake` or `aclocal`.

version specifies the particular tool revision to be used (see `devel/{automake,autoconf,libtool}[0-9]+` for valid versions).

operation is an optional extension to modify how the tool is used.

Multiple tools can be specified at once, either by including them all on a single line, or using the += Makefile construct.

Finally, there is the special tool, called `autotools`, which is a convenience function to bring in all available versions of the autotools to allow for cross-development work. This can also be accomplished by installing the `devel/autotools` port.

6.5.2. `libtool`

Shared libraries using the GNU building framework usually use `libtool` to adjust the compilation and installation of shared libraries to match the specifics of the underlying operating system. The usual practice is to use copy of `libtool` bundled with the application. In case you need to use external `libtool` , you can use the version provided by The Ports Collection:

```
USE_AUTOTOOLS= libtool:version [:env]
```

With no additional operations, `libtool:version` tells the building framework to patch the configure script with the system-installed copy of `libtool` . The `GNU_CONFIGURE` is implied. Further, a number of make and shell variables will be assigned for onward use by the port. See `bsd.autotools.mk` for details.

With the `:env` operation, only the environment will be set up.

Finally, `LIBTOOLFLAGS` and `LIBTOOLFILES` can be optionally set to override the most likely arguments to, and files patched by, `libtool` . Most ports are unlikely to need this. See `bsd.autotools.mk` for further details.

6.5.3. libltdl

Some ports make use of the libltdl library package, which is part of the libtool suite. Use of this library does not automatically necessitate the use of libtool itself, so a separate construct is provided.

```
USE_AUTOTOOLS= libltdl:version
```

Currently, all this does is to bring in a LIB_DEPENDS on the appropriate libltdl port, and is provided as a convenience function to help eliminate any dependencies on the autotools ports outside of the USE_AUTOTOOLS framework. There are no optional operations for this tool.

6.5.4. autoconf and autoheader

Some ports do not contain a configure script, but do contain an autoconf template in the configure.ac file. You can use the following assignments to let autoconf create the configure script, and also have autoheader create template headers for use by the configure script.

```
USE_AUTOTOOLS= autoconf:version[:env]
```

and

```
USE_AUTOTOOLS= autoheader:version
```

which also implies the use of autoconf:version .

Similarly to libtool , the inclusion of the optional :env operation simply sets up the environment for further use. Without it, patching and reconfiguration of the port is carried out.

The additional optional variables AUTOCONF_ARGS and AUTOHEADER_ARGS can be overridden by the port Makefile if specifically requested. As with the libtool equivalents, most ports are unlikely to need this.

6.5.5. automake and aclocal

Some packages only contain Makefile.am files. These have to be converted into Makefile.in files using automake, and the further processed by configure to generate an actual Makefile.

Similarly, packages occasionally do not ship with included aclocal.m4 files, again required to build the software. This can be achieved with aclocal, which scans configure.ac or configure.in .

aclocal has a similar relationship to automake as autoheader does to autoconf, described in the previous section. aclocal implies the use of automake, thus we have:

```
USE_AUTOTOOLS= automake:version[:env]
```

and

```
USE_AUTOTOOLS= aclocal:version
```

which also implies the use of automake:version .

Similarly to libtool and autoconf, the inclusion of the optional :env operation simply sets up the environment for further use. Without it, reconfiguration of the port is carried out.

As with autoconf and autoheader, both automake and aclocal have optional argument variables, AUTOMAKE_ARGS and ACLOCAL_ARGS respectively, which may be overridden by the port Makefile if required.

6.6. Using GNU gettext

6.6.1. Basic Usage

If your port requires gettext, set USES= gettext , and your port will inherit a dependency on devel/gettext. Other values for gettext usage are listed in Section 15.1, "Values of USES".

A rather common case is a port using gettext and configure. Generally, GNU configure should be able to locate gettext automatically. If it ever fails to, hints at the location of gettext can be passed in CPPFLAGS and LDFLAGS as follows:

```
USES= gettext
CPPFLAGS+= -I${LOCALBASE}/include
LDFLAGS+= -L${LOCALBASE}/lib

GNU_CONFIGURE= yes
```

Of course, the code can be more compact if there are no more flags to pass to configure:

```
USES= gettext
GNU_CONFIGURE= yes
```

6.6.2. Optional Usage

Some software products allow for disabling NLS, e.g., through passing --disable-nls to configure. In that case, your port should use gettext conditionally, depending on the status of the NLS option. For ports of low to medium complexity, you can rely on the following idiom:

```
GNU_CONFIGURE=  yes

.include <bsd.port.options.mk>
```

```
.if ${PORT_OPTIONS:MNLS}
USES+=   gettext
PLIST_SUB+=  NLS=""
.else
CONFIGURE_ARGS+= --disable-nls
PLIST_SUB+=  NLS="@comment -"
.endif

.include <bsd.port.mk>
```

The next item on your to-do list is to arrange so that the message catalog files are included in the packing list conditionally. The `Makefile` part of this task is already provided by the idiom. It is explained in the section on advanced `pkg-plist` practices. In a nutshell, each occurrence of `%%NLS%%` in `pkg-plist` will be replaced by "`@comment` " if NLS is disabled, or by a null string if NLS is enabled. Consequently, the lines prefixed by `%%NLS%%` will become mere comments in the final packing list if NLS is off; otherwise the prefix will be just left out. All you need to do now is insert `%%NLS%%` before each path to a message catalog file in `pkg-plist` . For example:

```
%%NLS%%share/locale/fr/LC_MESSAGES/foobar.mo
%%NLS%%share/locale/no/LC_MESSAGES/foobar.mo
```

In high complexity cases, you may need to use more advanced techniques than the recipe given here, such as dynamic packing list generation.

6.6.3. Handling Message Catalog Directories

There is a point to note about installing message catalog files. The target directories for them, which reside under `LOCALBASE/share/locale` , should rarely be created and removed by a port. The most popular languages have their respective directories listed in `PORTSDIR/Templates/BSD.local.dist` . The directories for many other languages are governed by the devel/gettext port. Consult its `pkg-plist` and see whether the port is going to install a message catalog file for a unique language.

6.7. Using Perl

If `MASTER_SITES` is set to `MASTER_SITE_PERL_CPAN` , then the preferred value of `MASTER_SITE_SUBDIR` is the top-level hierarchy name. For example, the recommended value for `p5-Module-Name` is `Module`. The top-level hierarchy can be examined at cpan.org. This keeps the port working when the author of the module changes.

The exception to this rule is when the relevant directory does not exist or the distfile does not exist in that directory. In such case, using author's id as `MASTER_SITE_SUBDIR` is allowed.

All of the tunable knobs below accept either `YES` or a version string like `5.8.0+` . `YES` means that the port can be used with any of the supported Perl versions. If a port only works with

specific versions of Perl, it can be indicated with a version string, specifying a minimum version (e.g., `5.7.3+`), a maximum version (e.g., `5.8.0-`) or an exact version (e.g., `5.8.3`).

Table 6.5. Variables for Ports That Use Perl

Variable	Meaning
`USE_PERL5`	The port uses Perl 5 to build and run.
`USE_PERL5_BUILD`	The port uses Perl 5 to build.
`USE_PERL5_RUN`	The port uses Perl 5 to run.
`PERL`	The full path of the Perl 5 interpreter, either in the system or installed from a port, but without the version number. Use this if you need to replace "`#!`"lines in scripts.
`PERL_CONFIGURE`	Configure using Perl's MakeMaker. It implies `USE_PERL5` .
`PERL_MODBUILD`	Configure, build and install using Module::Build. It implies `PERL_CONFIGURE` .
Read only variables	Means
`PERL_VERSION`	The full version of Perl installed (e.g., `5.8.9`).
`PERL_LEVEL`	The installed Perl version as an integer of the form `MNNNPP` (e.g., `500809`).
`PERL_ARCH`	Where Perl stores architecture dependent libraries. Defaults to `${ARCH}-freebsd` .
`PERL_PORT`	Name of the Perl port that is installed (e.g., `perl5`).
`SITE_PERL`	Directory name where site specific Perl packages go. This value is added to `PLIST_SUB` .

Note

Ports of Perl modules which do not have an official website should link to `cpan.org` in the WWW line of `pkg-descr`. The preferred URL form is `http://search.cpan.org/dist/Module-Name/` (including the trailing slash).

Note

Do not use `${SITE_PERL}` in dependency declarations. Doing so assumes that `bsd.perl.mk` has been included, which is not always true. Ports depending on this port will have incorrect dependencies if this port's files move later in an upgrade. The right way to declare Perl module dependencies is shown in the example below.

Example 6.2. Perl Dependency Example

```
p5-IO-Tee>=0.64:${PORTSDIR}/devel/p5-IO-Tee
```

For Perl ports that install manual pages, the macro `PERL5_MAN`*x* (where *x* ranges from `1` to `9`) can be used inside `pkg-plist`. For example,

```
lib/perl5/5.14/man/man3/AnyEvent::I3.3.gz
```

can be replaced with

```
%%PERL5_MAN3%%/AnyEvent::I3.3.gz
```

6.8. Using X11

6.8.1. X.Org Components

The X11 implementation available in The Ports Collection is X.Org. If your application depends on X components, set `USE_XORG` to the list of required components. Available components, at the time of writing, are:

`bigreqsproto compositeproto damageproto dmx dmxproto dri2proto evieproto fixesproto fontcacheproto fontenc fontsproto fontutil glproto ice inputproto kbproto libfs oldx pciaccess pixman printproto randrproto recordproto renderproto resourceproto scrnsaverproto sm trapproto videoproto x11 xau xaw xaw6 xaw7 xbitmaps xcmiscproto xcomposite xcursor xdamage xdmcp xevie xext xextproto xf86bigfontproto xf86dgaproto xf86driproto xf86miscproto xf86rushproto xf86vidmodeproto xfixes xfont xfontcache xft xi xinerama xineramaproto xkbfile xkbui xmu xmuu xorg-server xp xpm xprintapputil xprintutil xproto xproxymngproto xrandr xrender xres xscrnsaver xt xtrans xtrap xtst xv xvmc xxf86dga xxf86misc xxf86vm .`

Always up-to-date list can be found in `/usr/ports/Mk/bsd.xorg.mk` .

The Mesa Project is an effort to provide free OpenGL implementation. You can specify a dependency on various components of this project with `USE_GL` variable. Valid options are: `glut, glu, glw, glew, gl` and `linux`. For backwards compatibility, the value of `yes` maps to `glu`.

Example 6.3. USE_XORG Example

```
USE_XORG= xrender xft xkbfile xt xaw
USE_GL=  glu
```

Table 6.6. Variables for Ports That Use X

`USES= imake`	The port uses `imake`.
`XMKMF`	Set to the path of `xmkmf` if not in the `PATH`. Defaults to `xmkmf -a`.

Example 6.4. Using X11-Related Variables

```
# Use some X11 libraries
USE_XORG= x11 xpm
```

6.8.2. Ports That Require Motif

If your port requires a Motif library, define `USES= motif` in the `Makefile`. Default Motif implementation is x11-toolkits/open-motif. Users can choose x11-toolkits/lesstif instead by setting `WANT_LESSTIF` variable.

The `MOTIFLIB` variable will be set by `bsd.port.mk` to reference the appropriate Motif library. Please patch the source of your port to use `${MOTIFLIB}` wherever the Motif library is referenced in the original `Makefile` or `Imakefile`.

There are two common cases:

- If the port refers to the Motif library as `-lXm` in its `Makefile` or `Imakefile`, simply substitute `${MOTIFLIB}` for it.
- If the port uses `XmClientLibs` in its `Imakefile`, change it to `${MOTIFLIB} ${XTOOLLIB} ${XLIB}`.

Note that `MOTIFLIB` (usually) expands to `-L/usr/local/lib -lXm` or `/usr/local/lib/libXm.a`, so there is no need to add `-L` or `-l` in front.

6.8.3. X11 Fonts

If your port installs fonts for the X Window System, put them in `LOCALBASE/lib/X11/fonts/local`.

6.8.4. Getting a Fake `DISPLAY` with Xvfb

Some applications require a working X11 display for compilation to succeed. This pose a problem for machines that operate headless. When the following variable is used, the build infrastructure will start the virtual framebuffer X server. The working `DISPLAY` is then passed to the build.

```
USES= display
```

6.8.5. Desktop Entries

Desktop entries (a Freedesktop standard) provide a way to automatically adjust desktop features when a new program is installed, without requiring user intervention. For example, newly-installed programs automatically appear in the application menus of compatible desktop environments. Desktop entries originated in the GNOME desktop environment, but are now a standard and also work with KDE and Xfce. This bit of automation provides a real benefit to the user, and desktop entries are encouraged for applications which can be used in a desktop environment.

6.8.5.1. Using Predefined `.desktop` Files

Ports that include predefined `*.desktop` files should include those files in `pkg-plist` and install them in the `$LOCALBASE/share/applications` directory. The `INSTALL_DATA` macro is useful for installing these files.

6.8.5.2. Updating Desktop Database

If a port has a MimeType entry in its `portname.desktop`, the desktop database must be updated after install and deinstall. To do this, define `USES`= desktop-file-utils.

6.8.5.3. Creating Desktop Entries with the `DESKTOP_ENTRIES` Macro

Desktop entries can be easily created for applications by using the `DESKTOP_ENTRIES` variable. A file named `name.desktop` will be created, installed, and added to the `pkg-plist` automatically. Syntax is:

```
DESKTOP_ENTRIES= "NAME" -"COMMENT" -"ICON" -"COMMAND" -"CATEGORY" ↺
StartupNotify
```

The list of possible categories is available on the Freedesktop website. `StartupNotify` indicates whether the application is compatible with *startup notifications*. These are typical-

ly a graphic indicator like a clock that appear at the mouse pointer, menu, or panel to give the user an indication when a program is starting. A program that is compatible with startup notifications clears the indicator after it has started. Programs that are not compatible with startup notifications would never clear the indicator (potentially confusing and infuriating the user), and should have `StartupNotify` set to `false` so the indicator is not shown at all.

Example:

```
DESKTOP_ENTRIES= "ToME" -"Roguelike game based on JRR Tolkien's ↺
work" \
   "${DATADIR}/xtra/graf/tome-128.png" \
   "tome --v --g" -"Application;Game;RolePlaying;" \
   false
```

6.9. Using GNOME

The FreeBSD/GNOME project uses its own set of variables to define which GNOME components a particular port uses. A comprehensive list of these variables exists within the FreeBSD/GNOME project's homepage.

6.10. Using Qt

6.10.1. Ports That Require Qt

Table 6.7. Variables for Ports That Use Qt

`USE_QT4`	Specify tool and library dependencies for ports that use Qt 4. See Qt 4 component selection for more details.
`QT_PREFIX`	Set to the path where Qt installed to (read-only variable).
`MOC`	Set to the path of `moc` (read-only variable). Default set according to `USE_QT_VER` value.
`QTCPPFLAGS`	Additional compiler flags passed via `CONFIGURE_ENV` for Qt toolkit. Default set according to `USE_QT_VER` .
`QTCFGLIBS`	Additional libraries for linking passed via `CONFIGURE_ENV` for Qt toolkit. Default set according to `USE_QT_VER` .
`QTNONSTANDARD`	Suppress modification of `CONFIGURE_ENV` , `CONFIGURE_ARGS` , `CPPFLAGS` and `MAKE_ENV` .

Table 6.8. Additional Variables for Ports That Use Qt 4.x

UIC	Set to the path of uic (read-only variable).
QMAKE	Set to the path of qmake (read-only variable).
QMAKESPEC	Set to the path of configuration file for qmake (read-only variable).
QMAKEFLAGS	Additional flags for qmake.
QT_INCDIR	Set to Qt 4 include directories (read-only variable).
QT_LIBDIR	Set to Qt 4 libraries path (read-only variable).
QT_PLUGINDIR	Set to Qt 4 plugins path (read-only variable).

When USE_QT4 is set, the following settings are deployed:

```
CONFIGURE_ARGS+= --with-qt-includes=${QT_INCDIR} \
   --with-qt-libraries=${QT_LIBDIR} \
   --with-extra-libs=${LOCALBASE}/lib \
   --with-extra-includes=${LOCALBASE}/include
CONFIGURE_ENV+= MOC="${MOC}" UIC="${UIC}" LIBS="${QTCFGLIBS}" \
  QMAKE="${QMAKE}" QMAKESPEC="${QMAKESPEC}" QTDIR="${QT_PREFIX}"
MAKE_ENV+= QMAKESPEC="${QMAKESPEC}"

PLIST_SUB+= QT_INCDIR_REL=${QT_INCDIR_REL} \
  QT_LIBDIR_REL=${QT_LIBDIR_REL} \
  QT_PLUGINDIR_REL=${QT_PLUGINDIR_REL}
```

6.10.2. Component Selection

Individual Qt 4 tool and library dependencies must be specified in the USE_QT4 variable. Every component can be suffixed by either _build or _run, the suffix indicating whether the component should be depended on at buildtime or runtime, respectively. If unsuffixed, the component will be depended on at both build- and runtime. Usually, library components should be specified unsuffixed, tool components should be specified with the _build suffix and plugin components should be specified with the _run suffix. The most commonly used components are listed below (all available components are listed in _USE_QT4_ALL in /usr/ports/Mk/bsd.qt.mk):

Table 6.9. Available Qt 4 Library Components

Name	Description
corelib	core library (can be omitted unless the port uses nothing but corelib)

Name	Description
`gui`	graphical user interface library
`network`	network library
`opengl`	OpenGL library
`qt3support`	Qt 3 compatibility library
`qtestlib`	unit testing library
`script`	script library
`sql`	SQL library
`xml`	XML library

You can determine which libraries the application depends on, by running `ldd` on the main executable after a successful compilation.

Table 6.10. Available Qt 4 Tool Components

Name	Description
`moc`	meta object compiler (needed for almost every Qt application at buildtime)
`qmake`	Makefile generator / build utility
`rcc`	resource compiler (needed if the application comes with `*.rc` or `*.qrc` files)
`uic`	user interface compiler (needed if the application comes with `*.ui` files created by Qt Designer - in practice, every Qt application with a GUI)

Table 6.11. Available Qt 4 Plugin Components

Name	Description
`iconengines`	SVG icon engine plugin (if the application ships SVG icons)
`imageformats`	imageformat plugins for GIF, JPEG, MNG and SVG (if the application ships image files)

Example 6.5. Selecting Qt 4 Components

In this example, the ported application uses the Qt 4 graphical user interface library, the Qt 4 core library, all of the Qt 4 code generation tools and Qt 4's Makefile

generator. Since the `gui` library implies a dependency on the core library, `corelib` does not need to be specified. The Qt 4 code generation tools `moc`, `uic` and `rcc`, as well as the Makefile generator `qmake` are only needed at buildtime, thus they are specified with the `_build` suffix:

```
USE_QT4= gui moc_build qmake_build rcc_build uic_build
```

6.10.3. Using `qmake`

Table 6.12. Variables for Ports That Use `qmake`

Variable	Means
`QMAKE_ARGS`	Port specific QMake flags to be passed to the `qmake` binary.
`QMAKE_ENV`	Environment variables to be set for the `qmake` binary. The default is `${CONFIGURE_ENV}` .
`QMAKE_PRO`	Name of the project `.pro` file. The default is empty (using autodetection).

If the application does not provide a `configure` script but a `.pro` file, you can use the following:

```
USES= qmake
USE_QT4= qmake_build
```

`USES=qmake` instructs the port to use `qmake` for configuring. Note that `USES=qmake` does not imply a dependency on Qt 4 `qmake`, thus `USE_QT4` has to be populated with the `qmake_build` component.

Qt applications often are written to be cross-platform and often X11/Unix is not the platform they are developed on, which in turn often leads to certain loose ends, like:

- *Missing additional include paths.* Many applications come with system tray icon support, but neglect to look for includes and/or libraries in the X11 directories. You can tell `qmake` to add directories to the include and library search paths via the command line, for example:

  ```
  QMAKE_ARGS+= INCLUDEPATH+=${LOCALBASE}/include \
   LIBS+=-L${LOCALBASE}/lib
  ```

- *Bogus installation paths.* Sometimes data such as icons or .desktop files are by default installed into directories which are not scanned by XDG-compatible applications. editors/texmaker is an example for this - look at `patch-texmaker.pro` in the `files` di-

rectory of that port for a template on how to remedy this directly in the `qmake` project file.

6.11. Using KDE

6.11.1. KDE 4 Variable Definitions

If your application depends on KDE 4.x, set `USE_KDE4` to the list of required components. `_build` and `_run` suffixes can be used to force components dependency type (e.g., `baseapps_run`). If no suffix is set, a default dependency type will be used. If you want to force both types, add the component twice with both suffixes (e.g., `automoc4_build automoc4_run`). The most commonly used components are listed below (up-to-date components are documented at the top of `/usr/ports/Mk/bsd.kde4.mk`):

Table 6.13. Available KDE 4 Components

Name	Description
`kdehier`	Hierarchy of common KDE directories
`kdelibs`	KDE Developer Platform
`kdeprefix`	If set, port will be installed into `${KDE4_PREFIX}` instead of `${LOCALBASE}`
`sharedmime`	MIME types database for KDE ports
`automoc4`	Automatic moc for Qt 4 packages
`akonadi`	Storage server for KDE-Pim
`soprano`	Qt 4 RDF framework
`strigi`	Desktop search daemon
`libkcddb`	KDE CDDB library
`libkcompactdisc`	KDE library for interfacing with audio CDs
`libkdeedu`	Libraries used by educational applications
`libkdcraw`	KDE LibRaw library
`libkexiv2`	KDE Exiv2 library
`libkipi`	KDE Image Plugin Interface
`libkonq`	Konqueror core library
`libksane`	KDE SANE ("Scanner Access Now Easy") library
`pimlibs`	KDE-Pim libraries
`kate`	Text editor framework

Name	Description
marble	Virtual globe
okular	Universal document viewer
korundum	KDE Ruby bindings
perlkde	KDE Perl bindings
pykde4	KDE Python bindings
pykdeuic4	PyKDE user interface compiler
smokekde	KDE SMOKE libraries

KDE 4.x ports are installed into KDE4_PREFIX , which is /usr/local/kde4 currently. This is achieved by specifying the kdeprefix component, which overrides the default PREFIX. The ports however respect any PREFIX set via MAKEFLAGS environment variable and/or make arguments.

Example 6.6. USE_KDE4 Example

This is a simple example for a KDE 4 port. USES= cmake:outsource instructs the port to utilize CMake, a configuration tool widely used by KDE 4 projects (see Section 6.4.4, "Using cmake" for detailed usage). USE_KDE4 brings dependency on KDE libraries and makes port using automoc4 at build stage. Required KDE components and other dependencies can be determined through configure log. USE_KDE4 does not imply USE_QT4 . If a port requires some Qt 4 components, they should be specified in USE_QT4 .

```
USES=  cmake:outsource
USE_KDE4= kdelibs kdeprefix automoc4
USE_QT4= moc_build qmake_build rcc_build uic_build
```

6.12. Using Java

6.12.1. Variable Definitions

If your port needs a Java™ Development Kit (JDK™) to either build, run or even extract the distfile, then it should define USE_JAVA .

There are several JDKs in the ports collection, from various vendors, and in several versions. If your port must use one of these versions, you can define which one. The most current version, and FreeBSD default is java/openjdk6.

Table 6.14. Variables Which May be Set by Ports That Use Java

Variable	Means
USE_JAVA	Should be defined for the remaining variables to have any effect.
JAVA_VERSION	List of space-separated suitable Java versions for the port. An optional "+" allows you to specify a range of versions (allowed values: 1.5[+] 1.6[+] 1.7[+]).
JAVA_OS	List of space-separated suitable JDK port operating systems for the port (allowed values: native linux).
JAVA_VENDOR	List of space-separated suitable JDK port vendors for the port (allowed values: freebsd bsdjava sun openjdk).
JAVA_BUILD	When set, it means that the selected JDK port should be added to the build dependencies of the port.
JAVA_RUN	When set, it means that the selected JDK port should be added to the run dependencies of the port.
JAVA_EXTRACT	When set, it means that the selected JDK port should be added to the extract dependencies of the port.

Below is the list of all settings a port will receive after setting USE_JAVA :

Table 6.15. Variables Provided to Ports That Use Java

Variable	Value
JAVA_PORT	The name of the JDK port (e.g., 'java/openjdk6').
JAVA_PORT_VERSION	The full version of the JDK port (e.g., '1.6.0'). If you only need the first two digits of this version number, use ${JAVA_PORT_VERSION:C/^([0-9])\.([0-9])(.*)$/\1.\2/} .
JAVA_PORT_OS	The operating system used by the JDK port (e.g., 'native').
JAVA_PORT_VENDOR	The vendor of the JDK port (e.g., 'openjdk').

Variable	Value
JAVA_PORT_OS_DESCRIPTION	Description of the operating system used by the JDK port (e.g., 'Native').
JAVA_PORT_VENDOR_DESCRIPTION	Description of the vendor of the JDK port (e.g., 'OpenJDK BSD Porting Team').
JAVA_HOME	Path to the installation directory of the JDK (e.g., '/usr/local/openjdk6').
JAVAC	Path to the Java compiler to use (e.g., '/usr/local/openjdk6/bin/javac').
JAR	Path to the jar tool to use (e.g., '/usr/local/openjdk6/bin/jar' or '/usr/local/bin/fastjar').
APPLETVIEWER	Path to the appletviewer utility (e.g., '/usr/local/openjdk6/bin/appletviewer').
JAVA	Path to the java executable. Use this for executing Java programs (e.g., '/usr/local/openjdk6/bin/java').
JAVADOC	Path to the javadoc utility program.
JAVAH	Path to the javah program.
JAVAP	Path to the javap program.
JAVA_KEYTOOL	Path to the keytool utility program.
JAVA_N2A	Path to the native2ascii tool.
JAVA_POLICYTOOL	Path to the policytool program.
JAVA_SERIALVER	Path to the serialver utility program.
RMIC	Path to the RMI stub/skeleton generator, rmic.
RMIREGISTRY	Path to the RMI registry program, rmiregistry.
RMID	Path to the RMI daemon program rmid.
JAVA_CLASSES	Path to the archive that contains the JDK class files, ${JAVA_HOME}/jre/lib/rt.jar.

You may use the java-debug make target to get information for debugging your port. It will display the value of many of the forecited variables.

Additionally, the following constants are defined so all Java ports may be installed in a consistent way:

Table 6.16. Constants Defined for Ports That Use Java

Constant	Value
JAVASHAREDIR	The base directory for everything related to Java. Default: ${PREFIX}/share/java .
JAVAJARDIR	The directory where JAR files should be installed. Default: ${JAVASHAREDIR}/classes.
JAVALIBDIR	The directory where JAR files installed by other ports are located. Default: ${LOCALBASE}/share/java/classes .

The related entries are defined in both PLIST_SUB (documented in Section 7.1, "Changing pkg-plist Based on Make Variables") and SUB_LIST .

6.12.2. Building with Ant

When the port is to be built using Apache Ant, it has to define USE_ANT . Ant is thus considered to be the sub-make command. When no do-build target is defined by the port, a default one will be set that simply runs Ant according to MAKE_ENV , MAKE_ARGS and ALL_TARGET . This is similar to the USES= gmake mechanism, which is documented in Section 6.4, "Building Mechanisms".

6.12.3. Best Practices

When porting a Java library, your port should install the JAR file(s) in ${JAVAJARDIR}, and everything else under ${JAVASHAREDIR}/${PORTNAME} (except for the documentation, see below). In order to reduce the packing file size, you may reference the JAR file(s) directly in the Makefile. Just use the following statement (where myport.jar is the name of the JAR file installed as part of the port):

```
PLIST_FILES+= %%JAVAJARDIR%%/myport.jar
```

When porting a Java application, the port usually installs everything under a single directory (including its JAR dependencies). The use of ${JAVASHAREDIR}/${PORTNAME} is strongly encouraged in this regard. It is up the porter to decide whether the port should install the additional JAR dependencies under this directory or directly use the already installed ones (from ${JAVAJARDIR}).

Regardless of the type of your port (library or application), the additional documentation should be installed in the same location as for any other port. The JavaDoc tool is known to produce a different set of files depending on the version of the JDK that is used. For ports that do not enforce the use of a particular JDK, it is therefore a complex task to specify

the packing list (`pkg-plist`). This is one reason why porters are strongly encouraged to use the `PORTDOCS` macro. Moreover, even if you can predict the set of files that will be generated by `javadoc`, the size of the resulting `pkg-plist` advocates for the use of `PORTDOCS`.

The default value for `DATADIR` is `${PREFIX}/share/${PORTNAME}`. It is a good idea to override `DATADIR` to `${JAVASHAREDIR}/${PORTNAME}` for Java ports. Indeed, `DATADIR` is automatically added to `PLIST_SUB` (documented in Section 7.1, "Changing `pkg-plist` Based on Make Variables") so you may use `%%DATADIR%%` directly in `pkg-plist`.

As for the choice of building Java ports from source or directly installing them from a binary distribution, there is no defined policy at the time of writing. However, people from the FreeBSD Java Project encourage porters to have their ports built from source whenever it is a trivial task.

All the features that have been presented in this section are implemented in `bsd.java.mk`. If you ever think that your port needs more sophisticated Java support, please first have a look at the bsd.java.mk SVN log as it usually takes some time to document the latest features. Then, if you think the support you are lacking would be beneficial to many other Java ports, feel free to discuss it on the FreeBSD Java Language mailing list.

Although there is a `java` category for PRs, it refers to the JDK porting effort from the FreeBSD Java project. Therefore, you should submit your Java port in the `ports` category as for any other port, unless the issue you are trying to resolve is related to either a JDK implementation or `bsd.java.mk`.

Similarly, there is a defined policy regarding the `CATEGORIES` of a Java port, which is detailed in Section 5.3, "Categorization".

6.13. Web Applications, Apache and PHP

6.13.1. Apache

Table 6.17. Variables for Ports That Use Apache

`USE_APACHE`	The port requires Apache. Possible values: `yes` (gets any version), `22`, `24`, `22-24`, `22+`, etc. The default APACHE version is `22`. More details are available in `ports/Mk/bsd.apache.mk` and at wiki.freebsd.org/Apache/.
`APXS`	Full path to the `apxs` binary. Can be overridden in your port.
`HTTPD`	Full path to the `httpd` binary. Can be overridden in your port.

APACHE_VERSION	The version of present Apache installation (read-only variable). This variable is only available after inclusion of bsd.port.pre.mk . Possible values: 22, 24.
APACHEMODDIR	Directory for Apache modules. This variable is automatically expanded in pkg-plist.
APACHEINCLUDEDIR	Directory for Apache headers. This variable is automatically expanded in pkg-plist .
APACHEETCDIR	Directory for Apache configuration files. This variable is automatically expanded in pkg-plist .

Table 6.18. Useful Variables for Porting Apache Modules

MODULENAME	Name of the module. Default value is PORTNAME. Example: mod_hello
SHORTMODNAME	Short name of the module. Automatically derived from MODULENAME, but can be overridden. Example: hello
AP_FAST_BUILD	Use apxs to compile and install the module.
AP_GENPLIST	Also automatically creates a pkg-plist .
AP_INC	Adds a directory to a header search path during compilation.
AP_LIB	Adds a directory to a library search path during compilation.
AP_EXTRAS	Additional flags to pass to apxs.

6.13.2. Web Applications

Web applications should be installed into PREFIX/www/appname . For your convenience, this path is available both in Makefile and in pkg-plist as WWWDIR , and the path relative to PREFIX is available in Makefile as WWWDIR_REL .

The user and group of web server process are available as WWWOWN and WWWGRP, in case you need to change the ownership of some files. The default values of both are www. If you want different values for your port, use WWWOWN?= myuser notation, to allow user to override it easily.

Do not depend on Apache unless the web app explicitly needs Apache. Respect that users may wish to run your web app on different web server than Apache.

6.13.3. PHP

Table 6.19. Variables for Ports That Use PHP

USE_PHP	The port requires PHP. The value yes adds a dependency on PHP. The list of required PHP extensions can be specified instead. Example: pcre xml gettext
DEFAULT_PHP_VER	Selects which major version of PHP will be installed as a dependency when no PHP is installed yet. Default is 5. Possible values: 4, 5
IGNORE_WITH_PHP	The port does not work with PHP of the given version. Possible values: 4, 5
USE_PHPIZE	The port will be built as a PHP extension.
USE_PHPEXT	The port will be treated as a PHP extension, including installation and registration in the extension registry.
USE_PHP_BUILD	Set PHP as a build dependency.
WANT_PHP_CLI	Want the CLI (command line) version of PHP.
WANT_PHP_CGI	Want the CGI version of PHP.
WANT_PHP_MOD	Want the Apache module version of PHP.
WANT_PHP_SCR	Want the CLI or the CGI version of PHP.
WANT_PHP_WEB	Want the Apache module or the CGI version of PHP.

6.13.4. PEAR Modules

Porting PEAR modules is a very simple process.

Use the variables FILES, TESTS, DATA, SQLS, SCRIPTFILES, DOCS and EXAMPLES to list the files you want to install. All listed files will be automatically installed into the appropriate locations and added to pkg-plist .

Include ${PORTSDIR}/devel/pear/bsd.pear.mk on the last line of the Makefile.

Example 6.7. Example Makefile for PEAR Class

```
PORTNAME=       Date
PORTVERSION= 1.4.3
```

```
CATEGORIES= devel www pear

MAINTAINER= example@domain.com
COMMENT= PEAR Date and Time Zone Classes

BUILD_DEPENDS= ${PEARDIR}/PEAR.php:${PORTSDIR}/devel/pear-PEAR
RUN_DEPENDS:= ${BUILD_DEPENDS}

FILES=  Date.php Date/Calc.php Date/Human.php Date/
Span.php     \
  Date/TimeZone.php
TESTS=  test_calc.php test_date_methods_span.php ↺
testunit.php   \
  testunit_date.php testunit_date_span.php wknotest.txt   \
  bug674.php bug727_1.php bug727_2.php bug727_3.php       \
  bug727_4.php bug967.php weeksinmonth_4_monday.txt       \
  weeksinmonth_4_sunday.txt weeksinmonth_rdm_monday.txt   \
  weeksinmonth_rdm_sunday.txt
DOCS=  TODO
_DOCSDIR= .

.include <bsd.port.pre.mk>
.include -"${PORTSDIR}/devel/pear/bsd.pear.mk"
.include <bsd.port.post.mk>
```

6.14. Using Python

The Ports Collection supports parallel installation of multiple Python versions. Ports should make sure to use a correct `python` interpreter, according to the user-settable `PYTHON_VERSION` variable. Most prominently, this means replacing the path to `python` executable in scripts with the value of `PYTHON_CMD` variable.

Ports that install files under `PYTHON_SITELIBDIR` should use the `pyXY-` package name prefix, so their package name embeds the version of Python they are installed into.

```
PKGNAMEPREFIX= ${PYTHON_PKGNAMEPREFIX}
```

Table 6.20. Most Useful Variables for Ports That Use Python

`USE_PYTHON`	The port needs Python. Minimal required version can be specified with values such as `2.6+`. Version ranges can also be specified, by separating two version numbers with a dash, e.g.: `2.6-2.7`
`USE_PYDISTUTILS`	Use Python distutils for configuring, compiling and installing. This is required when

	the port comes with `setup.py`. This overrides the `do-build` and `do-install` targets and may also override `do-configure` if `GNU_CONFIGURE` is not defined.
`PYTHON_PKGNAMEPREFIX`	Used as a `PKGNAMEPREFIX` to distinguish packages for different Python versions. Example: `py24-`
`PYTHON_SITELIBDIR`	Location of the site-packages tree, that contains installation path of Python (usually `LOCALBASE`). The `PYTHON_SITELIBDIR` variable can be very useful when installing Python modules.
`PYTHONPREFIX_SITELIBDIR`	The PREFIX-clean variant of PYTHON_SITELIBDIR. Always use `%%PYTHON_SITELIBDIR%%` in `pkg-plist` when possible. The default value of `%%PYTHON_SITELIBDIR%%` is `lib/python%%PYTHON_VERSION%%/site-packages`
`PYTHON_CMD`	Python interpreter command line, including version number.
`PYNUMERIC`	Dependency line for numeric extension.
`PYNUMPY`	Dependency line for the new numeric extension, numpy. (PYNUMERIC is deprecated by upstream vendor).
`PYXML`	Dependency line for XML extension (not needed for Python 2.0 and higher as it is also in base distribution).
`USE_TWISTED`	Add dependency on twistedCore. The list of required components can be specified as a value of this variable. Example: `web lore pair flow`
`USE_ZOPE`	Add dependency on Zope, a web application platform. Change Python dependency to Python 2.7. Set `ZOPEBASEDIR` containing a directory with Zope installation.

A complete list of available variables can be found in `/usr/ports/Mk/bsd.python.mk`.

Some Python applications claim to have `DESTDIR` support (which would be required for staging) but it is broken (Mailman up to 2.1.16, for instance). This can be worked around by recompiling the scripts. This can be done, for example, in the `post-build` target. As-

suming the Python scripts are supposed to reside in PYTHONPREFIX_SITELIBDIR after installation, this solution can be applied:

```
(cd ${STAGEDIR}${PREFIX} \
  && ${PYTHON_CMD} ${PYTHON_LIBDIR}/compileall.py \
   -d ${PREFIX} -f ${PYTHONPREFIX_SITELIBDIR:S;${PREFIX}/;;})
```

This recompiles the sources with a path relative to the stage directory, and prepends the value of PREFIX to the file name recorded in the byte-compiled output file by -d. -f is required to force recompilation, and the :S;${PREFIX}/;; strips prefixes from the value of the PYTHONPREFIX_SITELIBDIR variable to make it relative to PREFIX.

Python 2.7 or newer is required for this. It does not work with Python 2.6.

6.15. Using Tcl/Tk

The Ports Collection supports parallel installation of multiple Tcl/Tk versions. Ports should try to support at least the default Tcl/Tk version and higher with the USE_TCL and USE_TK variables. It is possible to specify the desired version of tcl with the WITH_TCL_VER variable.

Table 6.21. The Most Useful Variables for Ports That Use Tcl/Tk

USE_TCL	The port depends on the Tcl library (not the shell). Minimal required version can be specified with values such as 84+. Individual unsupported versions can be specified with the INVALID_TCL_VER variable.
USE_TCL_BUILD	The port needs Tcl only during the build time.
USE_TCL_WRAPPER	Ports that require the Tcl shell and do not require a specific tclsh version should use this new variable. The tclsh wrapper is installed on the system. The user can specify the desired tcl shell to use.
WITH_TCL_VER	User-defined variable that sets the desired Tcl version.
UNIQUENAME _WITH_TCL_VER	Like WITH_TCL_VER , but per-port.
USE_TCL_THREADS	Require a threaded build of Tcl/Tk.
USE_TK	The port depends on the Tk library (not the wish shell). Implies USE_TCL with the same value. For more information see the description of USE_TCL variable.

USE_TK_BUILD	Analog to the USE_TCL_BUILD variable.
USE_TK_WRAPPER	Analog to the USE_TCL_WRAPPER variable.
WITH_TK_VER	Analog to the WITH_TCL_VER variable and implies WITH_TCL_VER of the same value.

A complete list of available variables can be found in /usr/ports/Mk/bsd.tcl.mk .

6.16. Using Emacs

This section is yet to be written.

6.17. Using Ruby

Table 6.22. Useful Variables for Ports That Use Ruby

Variable	Description
USE_RUBY	The port requires Ruby.
USE_RUBY_EXTCONF	The port uses extconf.rb to configure.
USE_RUBY_SETUP	The port uses setup.rb to configure.
RUBY_SETUP	Set to the alternative name of setup.rb. Common value is install.rb .

The following table shows the selected variables available to port authors via the ports infrastructure. These variables should be used to install files into their proper locations. Use them in pkg-plist as much as possible. These variables should not be redefined in the port.

Table 6.23. Selected Read-Only Variables for Ports That Use Ruby

Variable	Description	Example value
RUBY_PKGNAMEPREFIX	Used as a PKGNAMEPREFIX to distinguish packages for different Ruby versions.	ruby18-
RUBY_VERSION	Full version of Ruby in the form of x.y.z.	1.8.2
RUBY_SITELIBDIR	Architecture independent libraries installation path.	/usr/local/lib/ruby/site_ruby/1.8
RUBY_SITEARCHLIBDIR	Architecture dependent libraries installation path.	/usr/local/lib/ruby/site_ruby/1.8/amd64-freebsd6

Variable	Description	Example value
RUBY_MODDOCDIR	Module documentation installation path.	/usr/local/share/doc/ruby18/patsy
RUBY_MODEXAMPLESDIR	Module examples installation path.	/usr/local/share/examples/ruby18/patsy

A complete list of available variables can be found in /usr/ports/Mk/bsd.ruby.mk .

6.18. Using SDL

The USE_SDL variable is used to autoconfigure the dependencies for ports which use an SDL based library like devel/sdl12 and graphics/sdl_image.

The following SDL libraries for version 1.2 are recognized at the moment:

- sdl: devel/sdl12
- console: devel/sdl_console
- gfx: graphics/sdl_gfx
- image: graphics/sdl_image
- mixer: audio/sdl_mixer
- mm: devel/sdlmm
- net: net/sdl_net
- pango: x11-toolkits/sdl_pango
- sound: audio/sdl_sound
- ttf: graphics/sdl_ttf

The following SDL libraries for version 2.0 are recognized at the moment:

- sdl: devel/sdl20
- gfx: graphics/sdl2_gfx
- image: graphics/sdl2_image
- mixer: audio/sdl2_mixer
- net: net/sdl2_net
- ttf: graphics/sdl2_ttf

Therefore, if a port has a dependency on net/sdl_net and audio/sdl_mixer, the syntax will be:

```
USE_SDL= net mixer
```

The dependency devel/sdl12, which is required by net/sdl_net and audio/sdl_mixer, is automatically added as well.

If you use `USE_SDL` with entries using SDL 1.2, it will automatically:

- Add a dependency on sdl12-config to `BUILD_DEPENDS`
- Add the variable `SDL_CONFIG` to `CONFIGURE_ENV`
- Add the dependencies of the selected libraries to the `LIB_DEPENDS`

If you use `USE_SDL` with entries using SDL 2.0, it will automatically:

- Add a dependency on sdl2-config to `BUILD_DEPENDS`
- Add the variable `SDL2_CONFIG` to `CONFIGURE_ENV`
- Add the dependencies of the selected libraries to the `LIB_DEPENDS`

To check whether an SDL library is available, you can do it with the `WANT_SDL` variable:

```
WANT_SDL= yes

.include <bsd.port.pre.mk>

.if ${HAVE_SDL:Mmixer}!=""
USE_SDL+= mixer
.endif

.include <bsd.port.post.mk>
```

6.19. Using wxWidgets

This section describes the status of the wxWidgets libraries in the ports tree and its integration with the ports system.

6.19.1. Introduction

There are many versions of the wxWidgets libraries which conflict between them (install files under the same name). In the ports tree this problem has been solved by installing each version under a different name using version number suffixes.

The obvious disadvantage of this is that each application has to be modified to find the expected version. Fortunately, most of the applications call the `wx-config` script to determine the necessary compiler and linker flags. The script is named differently for every

available version. Majority of applications respect an environment variable, or accept a configure argument, to specify which `wx-config` script to call. Otherwise they have to be patched.

6.19.2. Version Selection

To make your port use a specific version of wxWidgets there are two variables available for defining (if only one is defined the other will be set to a default value):

Table 6.24. Variables to Select wxWidgets Versions

Variable	Description	Default value
`USE_WX`	List of versions the port can use	All available versions
`USE_WX_NOT`	List of versions the port can not use	None

The following is a list of available wxWidgets versions and the corresponding ports in the tree:

Table 6.25. Available wxWidgets Versions

Version	Port
`2.4`	x11-toolkits/wxgtk24
`2.6`	x11-toolkits/wxgtk26
`2.8`	x11-toolkits/wxgtk28

Note

The versions starting from `2.5` also come in Unicode version and are installed by a slave port named like the normal one plus a `-unicode` suffix, but this can be handled with variables (see Section 6.19.4, “Unicode”).

The variables in Table 6.24, “Variables to Select wxWidgets Versions” can be set to one or more of the following combinations separated by spaces:

Table 6.26. wxWidgets Version Specifications

Description	Example
Single version	`2.4`
Ascending range	`2.4+`

Description	Example
Descending range	`2.6-`
Full range (must be ascending)	`2.4-2.6`

There are also some variables to select the preferred versions from the available ones. They can be set to a list of versions, the first ones will have higher priority.

Table 6.27. Variables to Select Preferred wxWidgets Versions

Name	Designed for
`WANT_WX_VER`	the port
`WITH_WX_VER`	the user

6.19.3. Component Selection

There are other applications that, while not being wxWidgets libraries, are related to them. These applications can be specified in the `WX_COMPS` variable. The following components are available:

Table 6.28. Available wxWidgets Components

Name	Description	Version restriction
`wx`	main library	none
`contrib`	contributed libraries	none
`python`	wxPython (Python bindings)	`2.4-2.6`
`mozilla`	wxMozilla	`2.4`
`svg`	wxSVG	`2.6`

The dependency type can be selected for each component by adding a suffix separated by a semicolon. If not present then a default type will be used (see Table 6.30, “Default wxWidgets Dependency Types”). The following types are available:

Table 6.29. Available wxWidgets Dependency Types

Name	Description
`build`	Component is required for building, equivalent to `BUILD_DEPENDS`
`run`	Component is required for running, equivalent to `RUN_DEPENDS`
`lib`	Component is required for building and running, equivalent to `LIB_DEPENDS`

The default values for the components are detailed in the following table:

Table 6.30. Default wxWidgets Dependency Types

Component	Dependency type
wx	lib
contrib	lib
python	run
mozilla	lib
svg	lib

Example 6.8. Selecting wxWidgets Components

The following fragment corresponds to a port which uses wxWidgets version 2.4 and its contributed libraries.

```
USE_WX=  2.4
WX_COMPS= wx contrib
```

6.19.4. Unicode

The wxWidgets library supports Unicode since version 2.5. In the ports tree both versions are available and can be selected with the following variables:

Table 6.31. Variables to Select Unicode in wxWidgets Versions

Variable	Description	Designed for
WX_UNICODE	The port works *only* with the Unicode version	the port
WANT_UNICODE	The port works with both versions but prefers the Unicode one	the port
WITH_UNICODE	The port will use the Unicode version	the user
WITHOUT_UNICODE	The port will use the normal version if supported (when WX_UNICODE is not defined)	the user

Warning

Do not use `WX_UNICODE` for ports that can use both Unicode and normal versions. If you want the port to use Unicode by default define `WANT_UNICODE` instead.

6.19.5. Detecting Installed Versions

To detect an installed version you have to define `WANT_WX`. If you do not set it to a specific version then the components will have a version suffix. The `HAVE_WX` variable will be filled after detection.

Example 6.9. Detecting Installed wxWidgets Versions and Components

The following fragment can be used in a port that uses wxWidgets if it is installed, or an option is selected.

```
WANT_WX= yes

.include <bsd.port.pre.mk>

.if defined(WITH_WX) -|| -!empty(PORT_OPTIONS:MWX) -|| -!
empty(HAVE_WX:Mwx-2.4)
USE_WX=   2.4
CONFIGURE_ARGS+= --enable-wx
.endif
```

The following fragment can be used in a port that enables wxPython support if it is installed or if an option is selected, in addition to wxWidgets, both version `2.6`.

```
USE_WX=  2.6
WX_COMPS= wx
WANT_WX= 2.6

.include <bsd.port.pre.mk>

.if defined(WITH_WXPYTHON) -|| -!
empty(PORT_OPTIONS:MWXPYTHON) -|| -!empty(HAVE_WX:Mpython)
WX_COMPS+=  python
CONFIGURE_ARGS+= --enable-wxpython
```

```
.endif
```

6.19.6. Defined Variables

The following variables are available in the port (after defining one from Table 6.24, "Variables to Select wxWidgets Versions").

Table 6.32. Variables Defined for Ports That Use wxWidgets

Name	Description
WX_CONFIG	The path to the wxWidgets wx-config script (with different name)
WXRC_CMD	The path to the wxWidgets wxrc program (with different name)
WX_VERSION	The wxWidgets version that is going to be used (e.g., 2.6)
WX_UNICODE	If not defined but Unicode is going to be used then it will be defined

6.19.7. Processing in bsd.port.pre.mk

If you need to use the variables for running commands right after including bsd.port.pre.mk you need to define WX_PREMK.

Important

If you define WX_PREMK, then the version, dependencies, components and defined variables will not change if you modify the wxWidgets port variables *after* including bsd.port.pre.mk.

Example 6.10. Using wxWidgets Variables in Commands

The following fragment illustrates the use of WX_PREMK by running the wx-config script to obtain the full version string, assign it to a variable and pass it to the program.

```
USE_WX=  2.4
```

```
WX_PREMK= yes

.include <bsd.port.pre.mk>

.if exists(${WX_CONFIG})
VER_STR!= ${WX_CONFIG} ---release

PLIST_SUB+= VERSION="${VER_STR}"
.endif
```

Note

The wxWidgets variables can be safely used in commands when they are inside targets without the need of `WX_PREMK`.

6.19.8. Additional `configure` Arguments

Some GNU `configure` scripts can not find wxWidgets with just the `WX_CONFIG` environment variable set, requiring additional arguments. The `WX_CONF_ARGS` variable can be used for provide them.

Table 6.33. Legal Values for `WX_CONF_ARGS`

Possible value	Resulting argument
`absolute`	`--with-wx-config=${WX_CONFIG}`
`relative`	`--with-wx=${LOCALBASE} --with-wx-config=${WX_CONFIG:T}`

6.20. Using Lua

This section describes the status of the Lua libraries in the ports tree and its integration with the ports system.

6.20.1. Introduction

There are many versions of the Lua libraries and corresponding interpreters, which conflict between them (install files under the same name). In the ports tree this problem has been solved by installing each version under a different name using version number suffixes.

The obvious disadvantage of this is that each application has to be modified to find the expected version. But it can be solved by adding some additional flags to the compiler and linker.

6.20.2. Version Selection

To make your port use a specific version of Lua there are two variables available for defining (if only one is defined the other will be set to a default value):

Table 6.34. Variables to Select Lua Versions

Variable	Description	Default value
USE_LUA	List of versions the port can use	All available versions
USE_LUA_NOT	List of versions the port can not use	None

The following is a list of available Lua versions and the corresponding ports in the tree:

Table 6.35. Available Lua Versions

Version	Port
4.0	lang/lua4
5.0	lang/lua50
5.1	lang/lua

The variables in Table 6.34, "Variables to Select Lua Versions" can be set to one or more of the following combinations separated by spaces:

Table 6.36. Lua Version Specifications

Description	Example
Single version	4.0
Ascending range	5.0+
Descending range	5.0-
Full range (must be ascending)	5.0-5.1

There are also some variables to select the preferred versions from the available ones. They can be set to a list of versions, the first ones will have higher priority.

Table 6.37. Variables to Select Preferred Lua Versions

Name	Designed for
WANT_LUA_VER	the port

Name	Designed for
WITH_LUA_VER	the user

Example 6.11. Selecting the Lua Version

The following fragment is from a port which can use Lua version 5.0 or 5.1, and uses 5.0 by default. It can be overridden by the user with WITH_LUA_VER .

```
USE_LUA= 5.0-5.1
WANT_LUA_VER= 5.0
```

6.20.3. Component Selection

There are other applications that, while not being Lua libraries, are related to them. These applications can be specified in the LUA_COMPS variable. The following components are available:

Table 6.38. Available Lua Components

Name	Description	Version restriction
lua	main library	none
tolua	Library for accessing C/C++ code	4.0-5.0
ruby	Ruby bindings	4.0-5.0

Note

There are more components but they are modules for the interpreter, not used by applications (only by other modules).

The dependency type can be selected for each component by adding a suffix separated by a semicolon. If not present then a default type will be used (see Table 6.40, “Default Lua Dependency Types”). The following types are available:

Table 6.39. Available Lua Dependency Types

Name	Description
`build`	Component is required for building, equivalent to `BUILD_DEPENDS`
`run`	Component is required for running, equivalent to `RUN_DEPENDS`
`lib`	Component is required for building and running, equivalent to `LIB_DEPENDS`

The default values for the components are detailed in the following table:

Table 6.40. Default Lua Dependency Types

Component	Dependency type
`lua`	`lib` for `4.0-5.0` (shared) and `build` for `5.1` (static)
`tolua`	`build` (static)
`ruby`	`lib` (shared)

Example 6.12. Selecting Lua Components

The following fragment corresponds to a port which uses Lua version `4.0` and its Ruby bindings.

```
USE_LUA= 4.0
LUA_COMPS= lua ruby
```

6.20.4. Detecting Installed Versions

To detect an installed version you have to define `WANT_LUA` . If you do not set it to a specific version then the components will have a version suffix. The `HAVE_LUA` variable will be filled after detection.

Example 6.13. Detecting Installed Lua Versions and Components

The following fragment can be used in a port that uses Lua if it is installed, or an option is selected.

```
WANT_LUA= yes

.include <bsd.port.pre.mk>

.if defined(WITH_LUA5) -|| -!empty(PORT_OPTIONS:MLUA5) -|| -!
empty(HAVE_LUA:Mlua-5.[01])
USE_LUA=  5.0-5.1
CONFIGURE_ARGS+= --enable-lua5
.endif
```

The following fragment can be used in a port that enables tolua support if it is installed or if an option is selected, in addition to Lua, both version `4.0`.

```
USE_LUA= 4.0
LUA_COMPS= lua
WANT_LUA= 4.0

.include <bsd.port.pre.mk>

.if defined(WITH_TOLUA) -|| -!empty(PORT_OPTIONS:MTOLUA) -|| -!
empty(HAVE_LUA:Mtolua)
LUA_COMPS+=  tolua
CONFIGURE_ARGS+= --enable-tolua
.endif
```

6.20.5. Defined Variables

The following variables are available in the port (after defining one from Table 6.34, "Variables to Select Lua Versions").

Table 6.41. Variables Defined for Ports That Use Lua

Name	Description
`LUA_VER`	The Lua version that is going to be used (e.g., `5.1`)
`LUA_VER_SH`	The Lua shared library major version (e.g., `1`)
`LUA_VER_STR`	The Lua version without the dots (e.g., `51`)
`LUA_PREFIX`	The prefix where Lua (and components) is installed
`LUA_SUBDIR`	The directory under `${PREFIX}/bin`, `${PREFIX}/share` and `${PREFIX}/lib` where Lua is installed
`LUA_INCDIR`	The directory where Lua and tolua header files are installed

Name	Description
LUA_LIBDIR	The directory where Lua and tolua libraries are installed
LUA_MODLIBDIR	The directory where Lua module libraries (.so) are installed
LUA_MODSHAREDIR	The directory where Lua modules (.lua) are installed
LUA_PKGNAMEPREFIX	The package name prefix used by Lua modules
LUA_CMD	The path to the Lua interpreter
LUAC_CMD	The path to the Lua compiler
TOLUA_CMD	The path to the tolua program

Example 6.14. Telling the Port Where to Find Lua

The following fragment shows how to tell a port that uses a configure script where the Lua header files and libraries are.

```
USE_LUA= 4.0
GNU_CONFIGURE= yes
CONFIGURE_ENV= CPPFLAGS="-I${LUA_INCDIR}" LDFLAGS="-L
${LUA_LIBDIR}"
```

6.20.6. Processing in bsd.port.pre.mk

If you need to use the variables for running commands right after including bsd.port.pre.mk you need to define LUA_PREMK .

Important

If you define LUA_PREMK , then the version, dependencies, components and defined variables will not change if you modify the Lua port variables *after* including bsd.port.pre.mk .

Example 6.15. Using Lua Variables in Commands

The following fragment illustrates the use of `LUA_PREMK` by running the Lua interpreter to obtain the full version string, assign it to a variable and pass it to the program.

```
USE_LUA= 5.0
LUA_PREMK= yes

.include <bsd.port.pre.mk>

.if exists(${LUA_CMD})
VER_STR!= ${LUA_CMD} --v

CFLAGS+= -DLUA_VERSION_STRING="${VER_STR}"
.endif
```

Note

The Lua variables can be safely used in commands when they are inside targets without the need of `LUA_PREMK`.

6.21. Using `iconv`

After 2013-10-08 (
<revnumber>254273</revnumber>
), FreeBSD 10-CURRENT and newer versions have a native `iconv` in the operating system. On earlier versions, converters/libiconv was used as a dependency.

For software that needs `iconv`, define `USES=iconv`. FreeBSD versions before 10-CURRENT on 2013-08-13 (
<revnumber>254273</revnumber>
) do not have a native `iconv`. On these earlier versions, a dependency on converters/libiconv will be added automatically.

When a port defines `USES=iconv`, these variables will be available:

Variable name	Purpose	Value before FreeBSD 10-CURRENT <revnumber>254273</revnumber> (2013-08-13)	Value after FreeBSD 10-CURRENT <revnumber>254273</revnumber> (2013-08-13)
ICONV_CMD	Directory where the iconv binary resides	${LOCALBASE}/bin/iconv	/usr/bin/iconv
ICONV_LIB	ld argument to link to libiconv (if needed)	-liconv	(empty)
ICONV_PREFIX	Directory where the iconv implementation resides (useful for configure scripts)	${LOCALBASE}	/usr
ICONV_CONFIGURE_ARG	Preconstructed configure argument for configure scripts	--with-libiconv-prefix=${LOCALBASE}	(empty)
ICONV_CONFIGURE_BASE	Preconstructed configure argument for configure scripts	--with-libiconv=${LOCALBASE}	(empty)

These two examples automatically populate the variables with the correct value for systems using converters/libiconv or the native iconv respectively:

Example 6.16. Simple iconv Usage

```
USES=  iconv
LDFLAGS+= -L${LOCALBASE}/lib ${ICONV_LIB}
```

Example 6.17. iconv Usage with configure

```
USES=  iconv
CONFIGURE_ARGS+=${ICONV_CONFIGURE_ARG}
```

As shown above, `ICONV_LIB` is empty when a native `iconv` is present. This can be used to detect the native `iconv` and respond appropriately.

Sometimes a program has an `ld` argument or search path hardcoded in a `Makefile` or configure script. This approach can be used to solve that problem:

Example 6.18. Fixing Hardcoded `-liconv`

```
USES=  iconv

post-patch:
 @${REINPLACE_CMD} --e -'s/-liconv/${ICONV_LIB}/' ${WRKSRC}/
Makefile
```

In some cases it is necessary to set alternate values or perform operations depending on whether there is a native `iconv`. `bsd.port.pre.mk` must be included before testing the value of `ICONV_LIB`:

Example 6.19. Checking for Native `iconv` Availability

```
USES=  iconv

.include <bsd.port.pre.mk>

post-patch:
.if empty(ICONV_LIB)
 # native iconv detected
 @${REINPLACE_CMD} --e -'s|iconv||' ${WRKSRC}/Config.sh
.endif

.include <bsd.port.post.mk>
```

6.22. Using Xfce

The `USE_XFCE` variable is used to autoconfigure the dependencies for ports which use an Xfce based library or application like x11-toolkits/libxfce4gui and x11-wm/xfce4-panel.

The following Xfce libraries and applications are recognized at the moment:

- libexo: x11/libexo
- libgui: x11-toolkits/libxfce4gui
- libutil: x11/libxfce4util
- libmcs: x11/libxfce4mcs
- mcsmanager: sysutils/xfce4-mcs-manager
- panel: x11-wm/xfce4-panel
- thunar: x11-fm/thunar
- wm: x11-wm/xfce4-wm
- xfdev: dev/xfce4-dev-tools

The following additional parameters are recognized:

- configenv: Use this if your port requires a special modified `CONFIGURE_ENV` to find its required libraries.

  ```
  -I${LOCALBASE}/include --L${LOCALBASE}/lib
  ```

  gets added to CPPFLAGS to `CONFIGURE_ENV` .

Therefore, if a port has a dependency on sysutils/xfce4-mcs-manager and requires the special CPPFLAGS in its configure environment, the syntax will be:

```
USE_XFCE= mcsmanager configenv
```

6.23. Using Mozilla

Table 6.42. Variables for Ports That Use Mozilla

`USE_GECKO`	Gecko backend the port can handle. Possible values: `libxul` (`libxul.so`), `seamonkey` (`libgtkembedmoz.so` , deprecated, should not be used any more).
`USE_FIREFOX`	The port requires Firefox as a runtime dependency. Possible values: `yes` (get default version), `40`, `36`, `35`. Default dependency is on version `40`.
`USE_FIREFOX_BUILD`	The port requires Firefox as a build-time dependency. Possible values: see

	USE_FIREFOX. This automatically sets USE_FIREFOX and assigns the same value.
USE_SEAMONKEY	The port requires SeaMonkey as a runtime dependency. Possible values: yes (get default version), 20, 11 (deprecated, should not be used any more). Default dependency is on version 20.
USE_SEAMONKEY_BUILD	The port requires SeaMonkey as a buildtime dependency. Possible values: see USE_SEAMONKEY. This automatically sets USE_SEAMONKEY and assigns the same value.
USE_THUNDERBIRD	The port requires Thunderbird as a runtime dependency. Possible values: yes (get default version), 31, 30 (deprecated, should not be used any more). Default dependency is on version 31.
USE_THUNDERBIRD_BUILD	The port requires Thunderbird as a buildtime dependency. Possible values: see USE_THUNDERBIRD. This automatically sets USE_THUNDERBIRD and assigns the same value.

A complete list of available variables can be found in /usr/ports/Mk/bsd.gecko.mk .

6.24. Using Databases

Table 6.43. Variables for Ports Using Databases

Variable	Means
USE_BDB	If variable is set to yes, add dependency on databases/db41 port. The variable may also be set to values: 40, 41, 42, 43, 44, 46, 47, 48, or 51. You can declare a range of acceptable values, USE_BDB=42+ will find the highest installed version, and fall back to 42 if nothing else is installed.
USE_MYSQL	If variable is set to yes, add dependency on databases/mysql55-client port. An associated variable, WANT_MYSQL_VER , may be set to values such as 323, 40, 41, 50, 51, 52, 55, or 60.

Variable	Means
USE_PGSQL	If set to yes, add dependency on databases/postgresql90-client port. An associated variable, WANT_PGSQL_VER , may be set to values such as 83, 84, 90, 91 or 92. You can declare a minimum or maximum value; WANT_PGSQL_VER = 90+ will cause the port to depend on a minimum version of 9.0.
USE_SQLITE	If variable is set to yes, add dependency on databases/sqlite3 port. The variable may also be set to values: 3, 2.

More details are available in bsd.database.mk.

6.25. Starting and Stopping Services (rc Scripts)

rc.d scripts are used to start services on system startup, and to give administrators a standard way of stopping, starting and restarting the service. Ports integrate into the system rc.d framework. Details on its usage can be found in the rc.d Handbook chapter. Detailed explanation of available commands is provided in rc(8) and rc.subr(8). Finally, there is an article on practical aspects of rc.d scripting.

One or more rc.d scripts can be installed:

```
USE_RC_SUBR= doormand
```

Scripts must be placed in the files subdirectory and a .in suffix must be added to their filename. Standard SUB_LIST expansions will be used for this file. Use of the %%PREFIX% % and %%LOCALBASE%% expansions is strongly encouraged as well. More on SUB_LIST in the relevant section.

Prior to FreeBSD 6.1-RELEASE, integration with rcorder(8) is available by using USE_RCORDER instead of USE_RC_SUBR . However, use of this method is not necessary unless the port has an option to install itself in the base, or the service needs to run prior to the FILESYSTEMS rc.d script in the base.

As of FreeBSD 6.1-RELEASE, local rc.d scripts (including those installed by ports) are included in the overall rcorder(8) of the base system.

Example simple rc.d script:

```
#!/bin/sh

# $FreeBSD$
#
```

```
# PROVIDE: doormand
# REQUIRE: LOGIN
# KEYWORD: shutdown
#
# Add the following lines to -/etc/rc.conf.local or -/etc/rc.conf
# to enable this service:
#
# doormand_enable (bool): Set to NO by default.
#    Set it to YES to enable doormand.
# doormand_config (path): Set to %%PREFIX%%/etc/doormand/doormand.cf
#    by default.

. -/etc/rc.subr

name=doormand
rcvar=doormand_enable

load_rc_config $name

: ${doormand_enable:="NO"}
: ${doormand_config="%%PREFIX%%/etc/doormand/doormand.cf"}

command=%%PREFIX%%/sbin/${name}
pidfile=/var/run/${name}.pid

command_args="-p $pidfile --f $doormand_config"

run_rc_command -"$1"
```

Unless there is a good reason to start the service earlier all ports scripts should use

```
REQUIRE: LOGIN
```

If the service runs as a particular user (other than root) this is mandatory.

```
KEYWORD: shutdown
```

is included in the script above because the mythical port we are using as an example starts a service, and should be shut down cleanly when the system shuts down. If the script is not starting a persistent service this is not necessary.

For optional configuration elements the "=" style of default variable assignment is preferable to the ":=" style here, since the former sets a default value only if the variable is unset, and the latter sets one if the variable is unset *or* null. A user might very well include something like

```
doormand_flags=""
```

in their `rc.conf.local` file, and a variable substitution using ":=" would inappropriately override the user's intention. The `_enable` variable is not optional, and should use the ":" for the default.

Note

No new scripts should be added with the `.sh` suffix.

6.25.1. Pre-Commit Checklist

Before contributing a port with an `rc.d` script, and more importantly, before committing one, please consult the following checklist to be sure that it is ready.

1. If this is a new file, does it have `.sh` in the file name? If so that should be changed to just `file.in` since new `rc.d` files may not end with that extension.

2. Does the file have a `$FreeBSD$` tag?

3. Do the name of the file (minus `.in`), the `PROVIDE` line, and `$name` all match? The file name matching `PROVIDE` makes debugging easier, especially for rcorder(8) issues. Matching the file name and `$name` makes it easier to figure out which variables are relevant in `rc.conf[.local]` . The latter is also what you might call "policy" for all new scripts, including those in the base system.

4. Is the `REQUIRE` line set to `LOGIN`? This is mandatory for scripts that run as a non-root user. If it runs as root, is there a good reason for it to run prior to `LOGIN`? If not, it should run there so that we can loosely group local scripts to a point in rcorder(8) after most everything in the base is already running.

5. Does the script start a persistent service? If so, it should have `KEYWORD: shutdown`.

6. Make sure there is no `KEYWORD: FreeBSD` present. This has not been necessary or desirable for years. It is also an indication that the new script was copy/pasted from an old script, so extra caution should be given to the review.

7. If the script uses an interpreted language like `perl`, `python`, or `ruby`, make certain that `command_interpreter` is set appropriately. Otherwise,

   ```
   # service name stop
   ```

   will probably not work properly. See service(8) for more information.

8. Have all occurrences of `/usr/local` been replaced with `%%PREFIX%%`?

9. Do the default variable assignments come after `load_rc_config` ?

10. Are there default assignments to empty strings? They should be removed, but double-check that the option is documented in the comments at the top of the file.

11. Are things that are set in variables actually used in the script?

12. Are options listed in the default *name*`_flags` things that are actually mandatory? If so, they should be in `command_args`. The `-d` option is a red flag (pardon the pun) here, since it is usually the option to "daemonize" the process, and therefore is actually mandatory.

13. The *name*`_flags` variable should never be included in `command_args` (and vice versa, although that error is less common).

14. Does the script execute any code unconditionally? This is frowned on. Usually these things can/should be dealt with through a `start_precmd`.

15. All boolean tests should utilize the `checkyesno` function. No hand-rolled tests for `[Yy][Ee][Ss]`, etc.

16. If there is a loop (for example, waiting for something to start) does it have a counter to terminate the loop? We do not want the boot to be stuck forever if there is an error.

17. Does the script create files or directories that need specific permissions, for example, a `pid` file that needs to be owned by the user that runs the process? Rather than the traditional touch(1)/chown(8)/chmod(1) routine, consider using install(1) with the proper command line arguments to do the whole procedure with one step.

6.26. Adding Users and Groups

Some ports require a certain user to be on the installed system. Choose a free UID from 50 to 999 and register it either in `ports/UIDs` (for users) or in `ports/GIDs` (for groups). Make sure you do not use a UID already used by the system or other ports.

Please include a patch against these two files when you require a new user or group to be created for your port.

Then you can use `USERS` and `GROUPS` variables in your `Makefile`, and the user will be automatically created when installing the port.

```
USERS= pulse
GROUPS= pulse pulse-access pulse-rt
```

The current list of reserved UIDs and GIDs can be found in `ports/UIDs` and `ports/GIDs`.

6.27. Ports That Rely on Kernel Sources

Some ports (such as kernel loadable modules) need the kernel source files so that the port can compile. Here is the correct way to determine if the user has them installed:

```
USES= kmod
```

Apart from this check, the `kmod` feature takes care of most items that these ports need to take into account.

Chapter 7. Advanced pkg-plist Practices

7.1. Changing pkg-plist Based on Make Variables

Some ports, particularly the p5- ports, need to change their pkg-plist depending on what options they are configured with (or version of perl, in the case of p5- ports). To make this easy, any instances in the pkg-plist of %%OSREL%% , %%PERL_VER%% , and %%PERL_VERSION%% will be substituted for appropriately. The value of %%OSREL%% is the numeric revision of the operating system (e.g., 4.9). %%PERL_VERSION%% and %%PERL_VER%% is the full version number of perl (e.g., 5.8.9). Several other %%VARS%% related to port's documentation files are described in the relevant section.

If you need to make other substitutions, you can set the PLIST_SUB variable with a list of VAR=VALUE pairs and instances of %%VAR%% will be substituted with *VALUE* in the pkg-plist.

For instance, if you have a port that installs many files in a version-specific subdirectory, you can put something like

```
OCTAVE_VERSION= 2.0.13
PLIST_SUB= OCTAVE_VERSION=${OCTAVE_VERSION}
```

in the Makefile and use %%OCTAVE_VERSION%% wherever the version shows up in pkg-plist. That way, when you upgrade the port, you will not have to change dozens (or in some cases, hundreds) of lines in the pkg-plist .

If your port installs files conditionally on the options set in the port, the usual way of handling it is prefixing the pkg-plist lines with a %%TAG%% and adding that TAG to the PLIST_SUB variable inside the Makefile with a special value of @comment, which makes package tools to ignore the line:

```
.if defined(WITH_X11)
PLIST_SUB+= X11=""
.else
PLIST_SUB+= X11="@comment -"
.endif
```

and in the pkg-plist :

```
%%X11%%bin/foo-gui
```

This substitution will be done between the pre-install and do-install targets, by reading from PLIST and writing to TMPPLIST (default: WRKDIR/.PLIST.mktmp). So if your port

builds PLIST on the fly, do so in or before pre-install. Also, if your port needs to edit the resulting file, do so in post-install to a file named TMPPLIST.

Another way of modifying a port's packing list is based on setting the variables PLIST_FILES, PLIST_DIRS, and PLIST_DIRSTRY. The value of each variable is regarded as a list of pathnames to write to TMPPLIST along with PLIST contents. Names listed in PLIST_FILES, PLIST_DIRS, and PLIST_DIRSTRY are subject to %%VAR%% substitution as described above. Except for that, names from PLIST_FILES will appear in the final packing list unchanged, while @dirrm and @dirrmtry will be prepended to names from PLIST_DIRS and PLIST_DIRSTRY, respectively. To take effect, PLIST_FILES, PLIST_DIRS, and PLIST_DIRSTRY must be set before TMPPLIST is written, i.e., in pre-install or earlier.

7.2. Empty Directories

7.2.1. Cleaning Up Empty Directories

Do make your ports remove empty directories when they are de-installed. This is usually accomplished by adding @dirrm lines for all directories that are specifically created by the port. You need to delete subdirectories before you can delete parent directories.

```
 -:
lib/X11/oneko/pixmaps/cat.xpm
lib/X11/oneko/sounds/cat.au
 -:
@dirrm lib/X11/oneko/pixmaps
@dirrm lib/X11/oneko/sounds
@dirrm lib/X11/oneko
```

However, sometimes @dirrm will give you errors because other ports share the same directory. You can use @dirrmtry to remove only empty directories without warning.

```
@dirrmtry share/doc/gimp
```

This will neither print any error messages nor cause pkg_delete(1) to exit abnormally even if ${PREFIX}/share/doc/gimp is not empty due to other ports installing some files in there.

7.2.2. Creating Empty Directories

Empty directories created during port installation need special attention. They will not get created when installing the package, because packages only store the files, and pkg_add(1) creates directories for them as needed. To make sure the empty directory is created when installing the package, add this line to pkg-plist above the corresponding @dirrm line:

```
@exec mkdir --p %D/share/foo/templates
```

7.3. Configuration Files

If your port installs configuration files to PREFIX/etc (or elsewhere) do *not* simply list them in the pkg-plist . That will cause pkg_delete(1) to remove the files carefully edited by the user, and a re-installation will wipe them out.

Instead, install sample file(s) with a filename.sample suffix. Then copy the sample file to the real configuration file name, if it does not already exist. On deinstall delete the configuration file, but only if it is identical to the .sample file. You need to handle this both in the port Makefile, and in the pkg-plist (for installation from the package).

Example of the Makefile part:

```
post-install:
 @if [ -! --f ${PREFIX}/etc/orbit.conf -]; then \
 ${CP} --p ${PREFIX}/etc/orbit.conf.sample ${STAGEDIR}${PREFIX}/etc/
orbit.conf -; \
 fi
```

For each configuration file, create the following three lines in pkg-plist :

```
@unexec if cmp --s %D/etc/orbit.conf.sample %D/etc/orbit.conf; then ↺
rm --f %D/etc/orbit.conf; fi
etc/orbit.conf.sample
@exec if [ -! --f %D/etc/orbit.conf -] -; then cp --p %D/%F %B/
orbit.conf; fi
```

The order of these lines is important. On deinstallation, the sample file is compared to the actual configuration file. If these files are identical, no changes have been made by the user and the actual file can be safely deleted. Because the sample file must still exist for the comparison, the @unexec line comes before the sample configuration file name. On installation, if an actual configuration file is not already present, the sample file is copied to the actual file. The sample file must be present before it can be copied, so the @exec line comes after the sample configuration file name.

To debug any issues, temporarily remove the -s flag to cmp(1) for more output.

See pkg_create(1) for more information on %D and related substitution markers.

If there is a very good reason not to install a working configuration file by default, leave the @exec line out of pkg-plist and add a message pointing out that the user must copy and edit the file before the software will work.

7.4. Dynamic Versus Static Package List

A *static package list* is a package list which is available in the Ports Collection either as a pkg-plist file (with or without variable substitution), or embedded into the Makefile via

`PLIST_FILES` , `PLIST_DIRS` , and `PLIST_DIRSTRY` . Even if the contents are auto-generated by a tool or a target in the Makefile *before* the inclusion into the Ports Collection by a committer, this is still considered a static list, since it is possible to examine it without having to download or compile the distfile.

A *dynamic package list* is a package list which is generated at the time the port is compiled based upon the files and directories which are installed. It is not possible to examine it before the source code of the ported application is downloaded and compiled, or after running a `make clean` .

While the use of dynamic package lists is not forbidden, maintainers should use static package lists wherever possible, as it enables users to grep(1) through available ports to discover, for example, which port installs a certain file. Dynamic lists should be primarily used for complex ports where the package list changes drastically based upon optional features of the port (and thus maintaining a static package list is infeasible), or ports which change the package list based upon the version of dependent software used (e.g., ports which generate docs with Javadoc).

7.5. Automated Package List Creation

First, make sure your port is almost complete, with only `pkg-plist` missing. You may then run `make makeplist` to generate a `pkg-plist` automatically. This file must be double checked for correctness.

User configuration files should be removed, or installed as `filename.sample` . The `info/dir` file should not be listed and appropriate `install-info` lines should be added as noted in the info files section. Any libraries installed by the port should be listed as specified in the shared libraries section.

Chapter 8. The pkg-* Files

There are some tricks we have not mentioned yet about the pkg-* files that come in handy sometimes.

8.1. pkg-message

If you need to display a message to the installer, you may place the message in pkg-message. This capability is often useful to display additional installation steps to be taken after a pkg_add(1) or to display licensing information.

When some lines about the build-time knobs or warnings have to be displayed, use ECHO_MSG. The pkg-message file is only for post-installation steps. Likewise, the distinction between ECHO_MSG and ECHO_CMD should be kept in mind. The former is for printing informational text to the screen, while the latter is for command pipelining:

```
update-etc-shells:
 @${ECHO_MSG} -"updating -/etc/shells"
 @${CP} -/etc/shells -/etc/shells.bak
 @( ${GREP} --v ${PREFIX}/bin/bash -/etc/shells.bak; \
  ${ECHO_CMD} ${PREFIX}/bin/bash) >/etc/shells
 @${RM} -/etc/shells.bak
```

Note

The pkg-message file does not need to be added to pkg-plist.

8.2. pkg-install

If your port needs to execute commands when the binary package is installed with pkg_add(1) you can do this via the pkg-install script. This script will automatically be added to the package, and will be run twice by pkg_add(1): the first time as ${SH} pkg-install ${PKGNAME} PRE-INSTALL and the second time as ${SH} pkg-install ${PKGNAME} POST-INSTALL. $2 can be tested to determine which mode the script is being run in. The PKG_PREFIX environmental variable will be set to the package installation directory. See pkg_add(1) for additional information.

Note

This script is not run automatically if you install the port with `make install`. If you are depending on it being run, you will have to explicitly call it from your port's `Makefile`, with a line like `PKG_PREFIX=${PREFIX} ${SH} ${PKGINSTALL} ${PKGNAME} PRE-INSTALL`.

8.3. pkg-deinstall

This script executes when a package is removed.

This script will be run twice by pkg_delete(1). The first time as `${SH} pkg-deinstall ${PKGNAME} DEINSTALL` and the second time as `${SH} pkg-deinstall ${PKGNAME} POST-DEINSTALL`.

8.4. Changing the Names of pkg-* Files

All the names of `pkg-*` files are defined using variables so you can change them in your `Makefile` if need be. This is especially useful when you are sharing the same `pkg-*` files among several ports or have to write to one of the above files (see writing to places other than `WRKDIR` for why it is a bad idea to write directly into the `pkg-*` subdirectory).

Here is a list of variable names and their default values. (`PKGDIR` defaults to `${MASTER-DIR}`.)

Variable	Default value
DESCR	${PKGDIR}/pkg-descr
PLIST	${PKGDIR}/pkg-plist
PKGINSTALL	${PKGDIR}/pkg-install
PKGDEINSTALL	${PKGDIR}/pkg-deinstall
PKGMESSAGE	${PKGDIR}/pkg-message

Please change these variables rather than overriding `PKG_ARGS`. If you change `PKG_ARGS`, those files will not correctly be installed in `/var/db/pkg` upon install from a port.

8.5. Making Use of SUB_FILES and SUB_LIST

The SUB_FILES and SUB_LIST variables are useful for dynamic values in port files, such as the installation PREFIX in pkg-message.

The SUB_FILES variable specifies a list of files to be automatically modified. Each *file* in the SUB_FILES list must have a corresponding file.in present in FILESDIR. A modified version will be created in WRKDIR. Files defined as a value of USE_RC_SUBR (or the deprecated USE_RCORDER) are automatically added to the SUB_FILES. For the files pkg-message, pkg-install, and pkg-deinstall, the corresponding Makefile variable is automatically set to point to the processed version.

The SUB_LIST variable is a list of VAR=VALUE pairs. For each pair %%VAR%% will get replaced with VALUE in each file listed in SUB_FILES. Several common pairs are automatically defined: PREFIX, LOCALBASE, DATADIR, DOCSDIR, EXAMPLESDIR, WWWDIR, and ETCDIR. Any line beginning with @comment will be deleted from resulting files after a variable substitution.

The following example will replace %%ARCH%% with the system architecture in a pkg-message:

```
SUB_FILES= pkg-message
SUB_LIST= ARCH=${ARCH}
```

Note that for this example, the pkg-message.in file must exist in FILESDIR.

Example of a good pkg-message.in:

```
Now it is time to configure this package.
Copy %%PREFIX%%/share/examples/putsy/%%ARCH%%.conf into your home ↺
directory
as -.putsy.conf and edit it.
```

Chapter 9. Testing Your Port

9.1. Running `make describe`

Several of the FreeBSD port maintenance tools, such as portupgrade(1), rely on a database called `/usr/ports/INDEX` which keeps track of such items as port dependencies. `INDEX` is created by the top-level `ports/Makefile` via `make index`, which descends into each port subdirectory and executes `make describe` there. Thus, if `make describe` fails in any port, no one can generate `INDEX`, and many people will quickly become unhappy.

Note

It is important to be able to generate this file no matter what options are present in `make.conf` , so please avoid doing things such as using `.error` statements when (for instance) a dependency is not satisfied. (See Section 12.15, "Avoid Use of the `.error` Construct".)

If `make describe` produces a string rather than an error message, you are probably safe. See `bsd.port.mk` for the meaning of the string produced.

Also note that running a recent version of `portlint` (as specified in the next section) will cause `make describe` to be run automatically.

9.2. Portlint

Do check your work with `portlint` before you submit or commit it. `portlint` warns you about many common errors, both functional and stylistic. For a new (or repocopied) port, `portlint -A` is the most thorough; for an existing port, `portlint -C` is sufficient.

Since `portlint` uses heuristics to try to figure out errors, it can produce false positive warnings. In addition, occasionally something that is flagged as a problem really cannot be done in any other way due to limitations in the ports framework. When in doubt, the best thing to do is ask on FreeBSD ports mailing list.

9.3. Port Tools

The ports-mgmt/porttools program is part of the Ports Collection.

`port` is the front-end script, which can help you simplify the testing job. Whenever you want to test a new port or update an existing one, you can use `port test` to test your port, including the `portlint` checking. This command also detects and lists any files that are not listed in `pkg-plist`. See the following example:

```
# port test -/usr/ports/net/csup
```

9.4. PREFIX and DESTDIR

PREFIX determines where the port will be installed. It defaults to `/usr/local`, but can be set by the user to a custom path like `/opt`. Your port must respect the value of this variable.

DESTDIR, if set by the user, determines the complete alternative environment, usually a jail or an installed system mounted somewhere other than `/`. A port will actually install into DESTDIR/PREFIX, and register with the package database in `DESTDIR/var/db/pkg`. As DESTDIR is handled automatically by the ports infrastructure with chroot(8), you do not need any modifications or any extra care to write DESTDIR-compliant ports.

The value of PREFIX will be set to LOCALBASE (defaulting to `/usr/local`). If `USE_LINUX_PREFIX` is set, PREFIX will be LINUXBASE (defaulting to `/compat/linux`).

Avoiding hard-coded `/usr/local` paths in the source makes the port much more flexible and able to cater to the needs of other sites. Often, this can be accomplished by simply replacing occurrences of `/usr/local` in the port's various Makefiles with `${PREFIX}`. This variable is automatically passed down to every stage of the build and install processes.

Make sure your application is not installing things in `/usr/local` instead of PREFIX. A quick test for such hard-coded paths is:

```
# make clean; make package PREFIX=/var/tmp/`make --V PORTNAME`
```

If anything is installed outside of PREFIX, the package creation process will complain that it cannot find the files.

In addition, it is worth checking the same with the stage directory support (see Section 6.1, “Staging”):

```
# make stage && make check-orphans && make package
```

These tests will not find hard-coded paths inside the port's files, nor will it verify that LOCALBASE is being used to correctly refer to files from other ports. The temporarily-installed port in `/var/tmp/`make -V PORTNAME`` should be tested for proper operation to make sure there are no problems with paths.

PREFIX should not be set explicitly in a port's Makefile. Users installing the port may have set PREFIX to a custom location, and the port should respect that setting.

Refer to programs and files from other ports with the variables mentioned above, not explicit pathnames. For instance, if your port requires a macro `PAGER` to have the full pathname of `less`, do not use a literal path of `/usr/local/bin/less` . Instead, use `${LOCALBASE}`:

```
-DPAGER=\"${LOCALBASE}/bin/less\"
```

The path with `LOCALBASE` is more likely to still work if the system administrator has moved the whole `/usr/local` tree somewhere else.

9.5. Tinderbox

If you are an avid ports contributor, you might want to take a look at Tinderbox. It is a powerful system for building and testing ports. You can install Tinderbox using ports-mgmt/tinderbox port. Be sure to read supplied documentation since the configuration is not trivial.

Visit the Tinderbox website for more details.

Chapter 10. Upgrading an Individual Port

When you notice that a port is out of date compared to the latest version from the original authors, you should first ensure that you have the latest port. You can find them in the `ports/ports-current` directory of the FreeBSD FTP mirror sites. However, if you are working with more than a few ports, you will probably find it easier to use Subversion or portsnap(8) to keep your whole ports collection up-to-date, as described in the Handbook. This will have the added benefit of tracking all the ports' dependencies.

The next step is to see if there is an update already pending. To do this, you have two options. There is a searchable interface to the FreeBSD Problem Report (PR) database (also known as `GNATS`). Select `ports` in the dropdown, and enter the name of the port.

However, sometimes people forget to put the name of the port into the Synopsis field in an unambiguous fashion. In that case, you can try the FreeBSD Ports Monitoring System (also known as `portsmon`). This system attempts to classify port PRs by portname. To search for PRs about a particular port, use the Overview of One Port.

If there is no pending PR, the next step is to send an email to the port's maintainer, as shown by `make maintainer`. That person may already be working on an upgrade, or have a reason to not upgrade the port right now (because of, for example, stability problems of the new version); you would not want to duplicate their work. Note that unmaintained ports are listed with a maintainer of `ports@FreeBSD.org`, which is just the general ports mailing list, so sending mail there probably will not help in this case.

If the maintainer asks you to do the upgrade or there is no maintainer, then you have a chance to help out FreeBSD by preparing the update yourself! Please do this by using the diff(1) command in the base system.

To create a suitable `diff` for a single patch, copy the file that needs patching to *`something.orig`*, save your changes to *`something`* and then create your patch:

```
% diff --u something.orig something > something.diff
```

Otherwise, you should either use the `svn diff` method (Section 10.1, "Using SVN to Make Patches") or copy the contents of the port to an entire different directory and use the result of the recursive diff(1) output of the new and old ports directories (e.g., if your modified port directory is called `superedit` and the original is in our tree as `superedit.bak`, then save the result of `diff -ruN superedit.bak superedit`). Either unified or context diff is fine, but port committers generally prefer unified diffs. Note the use of the `-N` option—this is the accepted way to force diff to properly deal with the case of new files being added or old files being deleted. Before sending us the diff, please examine the output to

make sure all the changes make sense. (In particular, make sure you first clean out the work directories with `make clean`).

To simplify common operations with patch files, you can use `/usr/ports/Tools/scripts/patchtool.py`. Before using it, please read `/usr/ports/Tools/scripts/README.patchtool`.

If the port is unmaintained, and you are actively using it yourself, please consider volunteering to become its maintainer. FreeBSD has over 4000 ports without maintainers, and this is an area where more volunteers are always needed. (For a detailed description of the responsibilities of maintainers, refer to the section in the Developer's Handbook.)

The best way to send us the diff is by including it via send-pr(1) (category `ports`). If you are maintaining the port, be sure to put `[maintainer update]` at the beginning of your synopsis line and set the "Class" of your PR to `maintainer-update`. Otherwise, the "Class" of your PR should be `change-request`. Please mention any added or deleted files in the message, as they have to be explicitly specified to svn(1) when doing a commit. If the diff is more than about 20KB, please compress and uuencode it; otherwise, just include it in the PR as is.

Before you send-pr(1), you should review the Writing the problem report section in the Problem Reports article; it contains far more information about how to write useful problem reports.

Important

If your upgrade is motivated by security concerns or a serious fault in the currently committed port, please notify the Ports Management Team <portmgr@FreeBSD.org> to request immediate rebuilding and redistribution of your port's package. Unsuspecting users of pkg_add(1) will otherwise continue to install the old version via `pkg_add -r` for several weeks.

Note

Once again, please use diff(1) and not shar(1) to send updates to existing ports! This helps ports committers understand exactly what is being changed.

Now that you have done all that, you will want to read about how to keep up-to-date in Chapter 14, *Keeping Up*.

10.1. Using SVN to Make Patches

If you can, please submit a svn(1) diff — they are easier to handle than diffs between "new and old" directories. Plus it is easier for you to see what you have changed and to update your diff if something is modified in the Ports Collection from when you started to work on it until you submit your changes, or if the committer asks you to fix something.

```
% cd ~/my_wrkdir ❶
% svn co https://svn0.us-west.FreeBSD.org/ports/head/dns/pdnsd ❷
% cd ~/my_wrkdir/pdnsd
```

❶ This can be anywhere you want, of course; building ports is not limited to within `/usr/ports/`.

❷ svn0.us-west.FreeBSD.org is a public SVN server. Select the closest mirror and verify the mirror server certificate from the list of Subversion mirror sites.

While in the working directory, make any changes that you would usually make to the port. If you add or remove a file, use `svn` to track these changes:

```
% svn add new_file
% svn remove deleted_file
```

Make sure that you check the port using the checklist in Section 3.4, "Testing the Port" and Section 3.5, "Checking Your Port with `portlint`".

```
% svn status
% svn update ❶
```

❶ This will try to merge the differences between your patch and current SVN; watch the output carefully. The letter in front of each file name indicates what was done with it. See Table 10.1, "SVN Update File Prefixes" for a complete list.

Table 10.1. SVN Update File Prefixes

U	The file was updated without problems.
G	The file was updated without problems (you will only see this when working against a remote repository).
M	The file had been modified, and was merged without conflicts.
C	The file had been modified, and was merged with conflicts.

If you get `C` as a result of `svn update` it means something changed in the SVN repository and svn(1) was not able to merge your local changes and those from the repository. It is always a good idea to inspect the changes anyway, since svn(1) does not know anything

about how a port should be, so it might (and probably will) merge things that do not make sense.

The last step is to make a unified diff(1) of the files against SVN:

```
% svn diff > -../`basename ${PWD}`.diff
```

Note

Any files that have been removed should be explicitly mentioned in the PR, because file removal may not be obvious to the committer.

Send your patch following the guidelines in Chapter 10, *Upgrading an Individual Port*.

10.2. The Files UPDATING and MOVED

If upgrading the port requires special steps like changing configuration files or running a specific program, you should document this in the file `/usr/ports/UPDATING`. The format of an entry in this file is as follows:

```
YYYYMMDD:
  AFFECTS: users of portcategory/portname
  AUTHOR: Your name <Your email address>

  Special instructions
```

If you are including exact portmaster or portupgrading instructions, please make sure to get the shell escaping right.

The `/usr/ports/MOVED` file is used to list moved or removed ports. Each line in the file is made up of the name of the port, where the port was moved to, when, and why. If the port was removed, the section detailing where it was moved to can be left blank. Each section must be separated by the `|` (pipe) character, like so:

```
old name|new name (blank for deleted)|date of move|reason
```

The date should be entered in the form `YYYY-MM-DD`. New entries should be added to the end of the file to keep it in chronological order.

If a port was removed but has since been restored, delete the line in this file that states that it was removed.

The changes can be validated with `Tools/scripts/MOVEDlint.awk`.

Chapter 11. Ports Security

11.1. Why Security is So Important

Bugs are occasionally introduced to the software. Arguably, the most dangerous of them are those opening security vulnerabilities. From the technical viewpoint, such vulnerabilities are to be closed by exterminating the bugs that caused them. However, the policies for handling mere bugs and security vulnerabilities are very different.

A typical small bug affects only those users who have enabled some combination of options triggering the bug. The developer will eventually release a patch followed by a new version of the software, free of the bug, but the majority of users will not take the trouble of upgrading immediately because the bug has never vexed them. A critical bug that may cause data loss represents a graver issue. Nevertheless, prudent users know that a lot of possible accidents, besides software bugs, are likely to lead to data loss, and so they make backups of important data; in addition, a critical bug will be discovered really soon.

A security vulnerability is all different. First, it may remain unnoticed for years because often it does not cause software malfunction. Second, a malicious party can use it to gain unauthorized access to a vulnerable system, to destroy or alter sensitive data; and in the worst case the user will not even notice the harm caused. Third, exposing a vulnerable system often assists attackers to break into other systems that could not be compromised otherwise. Therefore closing a vulnerability alone is not enough: the audience should be notified of it in most clear and comprehensive manner, which will allow to evaluate the danger and take appropriate actions.

11.2. Fixing Security Vulnerabilities

While on the subject of ports and packages, a security vulnerability may initially appear in the original distribution or in the port files. In the former case, the original software developer is likely to release a patch or a new version instantly, and you will only need to update the port promptly with respect to the author's fix. If the fix is delayed for some reason, you should either mark the port as `FORBIDDEN` or introduce a patch file of your own to the port. In the case of a vulnerable port, just fix the port as soon as possible. In either case, the standard procedure for submitting your change should be followed unless you have rights to commit it directly to the ports tree.

Important

Being a ports committer is not enough to commit to an arbitrary port. Remember that ports usually have maintainers, whom you should respect.

Please make sure that the port's revision is bumped as soon as the vulnerability has been closed. That is how the users who upgrade installed packages on a regular basis will see they need to run an update. Besides, a new package will be built and distributed over FTP and WWW mirrors, replacing the vulnerable one. `PORTREVISION` should be bumped unless `PORTVERSION` has changed in the course of correcting the vulnerability. That is you should bump `PORTREVISION` if you have added a patch file to the port, but you should not if you have updated the port to the latest software version and thus already touched `PORTVERSION`. Please refer to the corresponding section for more information.

11.3. Keeping the Community Informed

11.3.1. The VuXML Database

A very important and urgent step to take as early after a security vulnerability is discovered as possible is to notify the community of port users about the jeopardy. Such notification serves two purposes. First, should the danger be really severe it will be wise to apply an instant workaround. E.g., stop the affected network service or even deinstall the port completely until the vulnerability is closed. Second, a lot of users tend to upgrade installed packages only occasionally. They will know from the notification that they *must* update the package without delay as soon as a corrected version is available.

Given the huge number of ports in the tree a security advisory cannot be issued on each incident without creating a flood and losing the attention of the audience when it comes to really serious matters. Therefore security vulnerabilities found in ports are recorded in the FreeBSD VuXML database. The Security Officer Team members also monitor it for issues requiring their intervention.

If you have committer rights you can update the VuXML database by yourself. So you will both help the Security Officer Team and deliver the crucial information to the community earlier. However, if you are not a committer, or you believe you have found an exceptionally severe vulnerability please do not hesitate to contact the Security Officer Team directly as described on the FreeBSD Security Information page.

The VuXML database is an XML document. Its source file `vuln.xml` is kept right inside the port security/vuxml. Therefore the file's full pathname will be `PORTSDIR/securi-`

`ty/vuxml/vuln.xml` . Each time you discover a security vulnerability in a port please add an entry for it to that file. Until you are familiar with VuXML, the best thing you can do is to find an existing entry fitting your case, then copy it and use it as a template.

11.3.2. A Short Introduction to VuXML

The full-blown XML format is complex, and far beyond the scope of this book. However, to gain basic insight on the structure of a VuXML entry you need only the notion of tags. XML tag names are enclosed in angle brackets. Each opening <tag> must have a matching closing </tag>. Tags may be nested. If nesting, the inner tags must be closed before the outer ones. There is a hierarchy of tags, i.e., more complex rules of nesting them. This is similar to HTML. The major difference is that XML is eXtensible, i.e., based on defining custom tags. Due to its intrinsic structure XML puts otherwise amorphous data into shape. VuXML is particularly tailored to mark up descriptions of security vulnerabilities.

Now consider a realistic VuXML entry:

```
<vuln vid="f4bc80f4-da62-11d8-90ea-0004ac98a7b9"> ❶
  <topic>Several vulnerabilities found in Foo</topic> ❷
  <affects>
    <package>
      <name>foo</name> ❸
      <name>foo-devel</name>
      <name>ja-foo</name>
      <range><ge>1.6</ge><lt>1.9</lt></range> ❹
      <range><ge>2.*</ge><lt>2.4_1</lt></range>
      <range><eq>3.0b1</eq></range>
    </package>
    <package>
      <name>openfoo</name> ❺
      <range><lt>1.10_7</lt></range> ❻
      <range><ge>1.2,1</ge><lt>1.3_1,1</lt></range>
    </package>
  </affects>
  <description>
    <body xmlns="http://www.w3.org/1999/xhtml">
      <p>J. Random Hacker reports:</p> ❼
      <blockquote
        cite="http://j.r.hacker.com/advisories/1">
        <p>Several issues in the Foo software may be exploited
          via carefully crafted QUUX requests.  These requests will
          permit the injection of Bar code, mumble theft, and the
          readability of the Foo administrator account.</p>
      </blockquote>
    </body>
  </description>
  <references> ❽
    <freebsdsa>SA-10:75.foo</freebsdsa> ❾
    <freebsdpr>ports/987654</freebsdpr> ❿
    <cvename>CAN-2010-0201</cvename> ⓫
```

```
    <cvename>CAN-2010-0466</cvename>
    <bid>96298</bid> ⓬
    <certsa>CA-2010-99</certsa> ⓭
    <certvu>740169</certvu> ⓮
    <uscertsa>SA10-99A</uscertsa> ⓯
    <uscertta>SA10-99A</uscertta> ⓰
    <mlist msgid="201075606@hacker.com">http://
marc.theaimsgroup.com/?l=bugtraq&amp;m=203886607825605</mlist> ⓱
    <url>http://j.r.hacker.com/advisories/1</url> ⓲
  </references>
  <dates>
    <discovery>2010-05-25</discovery> ⓳
    <entry>2010-07-13</entry> ⓴
    <modified>2010-09-17</modified> ㉑
  </dates>
</vuln>
```

The tag names are supposed to be self-explanatory so we shall take a closer look only at fields you will need to fill in by yourself:

❶ This is the top-level tag of a VuXML entry. It has a mandatory attribute, `vid`, specifying a universally unique identifier (UUID) for this entry (in quotes). You should generate a UUID for each new VuXML entry (and do not forget to substitute it for the template UUID unless you are writing the entry from scratch). You can use uuidgen(1) to generate a VuXML UUID.

❷ This is a one-line description of the issue found.

❸ The names of packages affected are listed there. Multiple names can be given since several packages may be based on a single master port or software product. This may include stable and development branches, localized versions, and slave ports featuring different choices of important build-time configuration options.

Important

It is your responsibility to find all such related packages when writing a VuXML entry. Keep in mind that `make search name=foo` is your friend. The primary points to look for are as follows:

- the `foo-devel` variant for a `foo` port;
- other variants with a suffix like `-a4` (for print-related packages), `-without-gui` (for packages with X support disabled), or similar;

- `jp-`, `ru-`, `zh-`, and other possible localized variants in the corresponding national categories of the ports collection.

❹ Affected versions of the package(s) are specified there as one or more ranges using a combination of `<lt>`, `<le>`, `<eq>`, `<ge>`, and `<gt>` elements. The version ranges given should not overlap.

In a range specification, * (asterisk) denotes the smallest version number. In particular, `2.*` is less than `2.a`. Therefore an asterisk may be used for a range to match all possible `alpha`, `beta`, and `RC` versions. For instance, `<ge>2.*</ge><lt>3.*</lt>` will selectively match every `2.x` version while `<ge>2.0</ge><lt>3.0</lt>` will not since the latter misses `2.r3` and matches `3.b`.

The above example specifies that affected are versions from `1.6` to `1.9` inclusive, versions `2.x` before `2.4_1`, and version `3.0b1`.

❺ Several related package groups (essentially, ports) can be listed in the `<affected>` section. This can be used if several software products (say FooBar, FreeBar and OpenBar) grow from the same code base and still share its bugs and vulnerabilities. Note the difference from listing multiple names within a single <package> section.

❻ The version ranges should allow for `PORTEPOCH` and `PORTREVISION` if applicable. Please remember that according to the collation rules, a version with a non-zero `PORTEPOCH` is greater than any version without `PORTEPOCH`, e.g., `3.0,1` is greater than `3.1` or even than `8.9`.

❼ This is a summary of the issue. XHTML is used in this field. At least enclosing `<p>` and `</p>` should appear. More complex mark-up may be used, but only for the sake of accuracy and clarity: No eye candy please.

❽ This section contains references to relevant documents. As many references as apply are encouraged.

❾ This is a FreeBSD security advisory.

❿ This is a FreeBSD problem report.

⓫ This is a MITRE CVE identifier.

⓬ This is a SecurityFocus Bug ID.

⓭ This is a US-CERT security advisory.

⓮ This is a US-CERT vulnerability note.

⓯ This is a US-CERT Cyber Security Alert.

⓰ This is a US-CERT Technical Cyber Security Alert.

⓱ This is a URL to an archived posting in a mailing list. The attribute `msgid` is optional and may specify the message ID of the posting.

⓲ This is a generic URL. It should be used only if none of the other reference categories apply.

⓳ This is the date when the issue was disclosed (*YYYY-MM-DD*).

⓴ This is the date when the entry was added (*YYYY-MM-DD*).

21 This is the date when any information in the entry was last modified (*YYYY-MM-DD*). New entries must not include this field. It should be added upon editing an existing entry.

11.3.3. Testing Your Changes to the VuXML Database

Assume you just wrote or filled in an entry for a vulnerability in the package `clamav` that has been fixed in version `0.65_7`.

As a prerequisite, you need to *install* fresh versions of the ports ports-mgmt/portaudit, ports-mgmt/portaudit-db, and security/vuxml.

Note

To run `packaudit` you must have permission to write to its `DATABASEDIR`, typically `/var/db/portaudit` .

To use a different directory set the `DATABASEDIR` environment variable to a different location.

If you are working in a directory other than `${PORTSDIR}/security/vuxml` set the `VUXMLDIR` environment variable to the directory where `vuln.xml` is located.

First, check whether there already is an entry for this vulnerability. If there were such an entry, it would match the previous version of the package, `0.65_6`:

```
% packaudit
% portaudit clamav-0.65_6
```

If there is none found, you have the green light to add a new entry for this vulnerability.

```
% cd ${PORTSDIR}/security/vuxml
% make newentry
```

When you are done verify its syntax and formatting.

```
% make validate
```

Note

You will need at least one of the following packages installed: textproc/libxml2, textproc/jade.

Now rebuild the `portaudit` database from the VuXML file:

```
% packaudit
```

To verify that the `<affected>` section of your entry will match correct package(s), issue the following command:

```
% portaudit --f -/usr/ports/INDEX --r uuid
```

Note

Please refer to portaudit(1) for better understanding of the command syntax.

Make sure that your entry produces no spurious matches in the output.

Now check whether the right package versions are matched by your entry:

```
% portaudit clamav-0.65_6 clamav-0.65_7
Affected package: clamav-0.65_6 (matched by clamav<0.65_7)
Type of problem: clamav remote denial-of-service.
Reference: <http://www.freebsd.org/ports/
portaudit/74a9541d-5d6c-11d8-80e3-0020ed76ef5a.html>

1 problem(s) found.
```

The former version should match while the latter one should not.

Finally, verify whether the web page generated from the VuXML database looks like expected:

```
% mkdir --p ~/public_html/portaudit
% packaudit
% lynx ~/public_html/
portaudit/74a9541d-5d6c-11d8-80e3-0020ed76ef5a.html
```

Chapter 12. Dos and Don'ts

12.1. Introduction

Here is a list of common dos and don'ts that you encounter during the porting process. You should check your own port against this list, but you can also check ports in the PR database that others have submitted. Submit any comments on ports you check as described in Bug Reports and General Commentary. Checking ports in the PR database will both make it faster for us to commit them, and prove that you know what you are doing.

12.2. `WRKDIR`

Do not write anything to files outside `WRKDIR`. `WRKDIR` is the only place that is guaranteed to be writable during the port build (see installing ports from a CDROM for an example of building ports from a read-only tree). If you need to modify one of the `pkg-*` files, do so by redefining a variable, not by writing over it.

12.3. `WRKDIRPREFIX`

Make sure your port honors `WRKDIRPREFIX`. Most ports do not have to worry about this. In particular, if you are referring to a `WRKDIR` of another port, note that the correct location is `WRKDIRPREFIXPORTSDIR/subdir/name/work` not `PORTSDIR/subdir/name/work` or `.CURDIR/../../subdir/name/work` or some such.

Also, if you are defining `WRKDIR` yourself, make sure you prepend `${WRKDIRPREFIX}${.CURDIR}` in the front.

12.4. Differentiating Operating Systems and OS Versions

You may come across code that needs modifications or conditional compilation based upon what version of FreeBSD Unix it is running under. The preferred way to tell FreeBSD versions apart are the `__FreeBSD_version` and `__FreeBSD__` macros defined in sys/param.h. If this file is not included add the code,

```
#include <sys/param.h>
```

to the proper place in the `.c` file.

__FreeBSD__ is defined in all versions of FreeBSD as their major version number. For example, in FreeBSD 9.x, __FreeBSD__ is defined to be 9.

```
#if __FreeBSD__ >= 9
#  if __FreeBSD_version >= 901000
  -/* 9.1+ release specific code here */
#  endif
#endif
```

12.5. Writing Something After bsd.port.mk

Do not write anything after the .include <bsd.port.mk> line. It usually can be avoided by including bsd.port.pre.mk somewhere in the middle of your Makefile and bsd.port.post.mk at the end.

Note

Include either the bsd.port.pre.mk /bsd.port.post.mk pair or bsd.port.mk only; do not mix these two usages.

bsd.port.pre.mk only defines a few variables, which can be used in tests in the Makefile, bsd.port.post.mk defines the rest.

Here are some important variables defined in bsd.port.pre.mk (this is not the complete list, please read bsd.port.mk for the complete list).

Variable	Description
ARCH	The architecture as returned by uname -m (e.g., i386)
OPSYS	The operating system type, as returned by uname -s (e.g., FreeBSD)
OSREL	The release version of the operating system (e.g., 2.1.5 or 2.2.7)
OSVERSION	The numeric version of the operating system; the same as __FreeBSD_version .
LOCALBASE	The base of the “local” tree (e.g., /usr/local)
PREFIX	Where the port installs itself (see more on PREFIX).

Note

If you have to define the variable `MASTERDIR`, do so before including `bsd.port.pre.mk` .

Here are some examples of things you can write after `bsd.port.pre.mk` :

```
# no need to compile lang/perl5 if perl5 is already in system
.if ${OSVERSION} > 300003
BROKEN= perl is in system
.endif
```

You did remember to use tab instead of spaces after `BROKEN=` and :-).

12.6. Use the `exec` Statement in Wrapper Scripts

If the port installs a shell script whose purpose is to launch another program, and if launching that program is the last action performed by the script, make sure to launch the program using the `exec` statement, for instance:

```
#!/bin/sh
exec %%LOCALBASE%%/bin/java --jar %%DATADIR%%/foo.jar -"$@"
```

The `exec` statement replaces the shell process with the specified program. If `exec` is omitted, the shell process remains in memory while the program is executing, and needlessly consumes system resources.

12.7. Do Things Rationally

The `Makefile` should do things simply and reasonably. If you can make it a couple of lines shorter or more readable, then do so. Examples include using a make `.if` construct instead of a shell `if` construct, not redefining `do-extract` if you can redefine `EXTRACT*` instead, and using `GNU_CONFIGURE` instead of `CONFIGURE_ARGS += --prefix=${PREFIX}` .

If you find yourself having to write a lot of new code to try to do something, please go back and review `bsd.port.mk` to see if it contains an existing implementation of what you are trying to do. While hard to read, there are a great many seemingly-hard problems for which `bsd.port.mk` already provides a shorthand solution.

12.8. Respect Both CC and CXX

The port must respect both CC and CXX variables. What we mean by this is that the port must not set the values of these variables absolutely, overriding existing values; instead, it may append whatever values it needs to the existing values. This is so that build options that affect all ports can be set globally.

If the port does not respect these variables, please add NO_PACKAGE=ignores either cc or cxx to the Makefile.

An example of a Makefile respecting both CC and CXX variables follows. Note the ?=:

```
CC?= gcc
```

```
CXX?= g++
```

Here is an example which respects neither CC nor CXX variables:

```
CC= gcc
```

```
CXX= g++
```

Both CC and CXX variables can be defined on FreeBSD systems in /etc/make.conf . The first example defines a value if it was not previously set in /etc/make.conf , preserving any system-wide definitions. The second example clobbers anything previously defined.

12.9. Respect CFLAGS

The port must respect the CFLAGS variable. What we mean by this is that the port must not set the value of this variable absolutely, overriding the existing value; instead, it may append whatever values it needs to the existing value. This is so that build options that affect all ports can be set globally.

If it does not, please add NO_PACKAGE=ignores cflags to the Makefile.

An example of a Makefile respecting the CFLAGS variable follows. Note the +=:

```
CFLAGS+= --Wall --Werror
```

Here is an example which does not respect the CFLAGS variable:

```
CFLAGS= --Wall --Werror
```

The CFLAGS variable is defined on FreeBSD systems in /etc/make.conf . The first example appends additional flags to the CFLAGS variable, preserving any system-wide definitions. The second example clobbers anything previously defined.

You should remove optimization flags from the third party `Makefiles`. System `CFLAGS` contains system-wide optimization flags. An example from an unmodified `Makefile`:

```
CFLAGS= --O3 --funroll-loops --DHAVE_SOUND
```

Using system optimization flags, the `Makefile` would look similar to the following example:

```
CFLAGS+= --DHAVE_SOUND
```

12.10. Threading Libraries

The threading library must be linked to the binaries using a special flag `-pthread` on FreeBSD. If a port insists on linking `-lpthread` directly, patch it to use `-pthread`.

Note

If building the port errors out with `unrecognized option '-pthread'`, it may be desirable to use `cc` as linker by setting `CONFIGURE_ENV` to `LD=${CC}`. The `-pthread` option is not supported by `ld` directly.

12.11. Feedback

Do send applicable changes/patches to the original author/maintainer for inclusion in next release of the code. This will only make your job that much easier for the next release.

12.12. `README.html`

Do not include the `README.html` file. This file is not part of the SVN collection but is generated using the `make readme` command.

Note

If `make readme` fails, make sure that the default value of `ECHO_MSG` has not been modified by the port.

12.13. Marking a Port Not Installable with BROKEN, FORBIDDEN, or IGNORE

In certain cases users should be prevented from installing a port. To tell a user that a port should not be installed, there are several `make` variables that can be used in a port's `Makefile`. The value of the following `make` variables will be the reason that is given back to users for why the port refuses to install itself. Please use the correct `make` variable as each make variable conveys radically different meanings to both users, and to automated systems that depend on the `Makefiles`, such as the ports build cluster, FreshPorts, and portsmon.

12.13.1. Variables

- `BROKEN` is reserved for ports that currently do not compile, install, or deinstall correctly. It should be used for ports where the problem is believed to be temporary.

  If instructed, the build cluster will still attempt to try to build them to see if the underlying problem has been resolved. (However, in general, the cluster is run without this.)

  For instance, use `BROKEN` when a port:

  - does not compile
  - fails its configuration or installation process
  - installs files outside of `${LOCALBASE}`
  - does not remove all its files cleanly upon deinstall (however, it may be acceptable, and desirable, for the port to leave user-modified files behind)

- `FORBIDDEN` is used for ports that contain a security vulnerability or induce grave concern regarding the security of a FreeBSD system with a given port installed (e.g., a reputably insecure program or a program that provides easily exploitable services). Ports should be marked as `FORBIDDEN` as soon as a particular piece of software has a vulnerability and there is no released upgrade. Ideally ports should be upgraded as soon as possible when a security vulnerability is discovered so as to reduce the number of vulnerable FreeBSD hosts (we like being known for being secure), however sometimes there is a noticeable time gap between disclosure of a vulnerability and an updated release of the vulnerable software. Do not mark a port `FORBIDDEN` for any reason other than security.

- `IGNORE` is reserved for ports that should not be built for some other reason. It should be used for ports where the problem is believed to be structural. The build cluster will not, under any circumstances, build ports marked as `IGNORE`. For instance, use `IGNORE` when a port:

  - compiles but does not run properly

- does not work on the installed version of FreeBSD
- has a distfile which may not be automatically fetched due to licensing restrictions
- does not work with some other currently installed port (for instance, the port depends on www/apache20 but www/apache22 is installed)

Note

If a port would conflict with a currently installed port (for example, if they install a file in the same place that performs a different function), use `CONFLICTS` instead. `CONFLICTS` will set `IGNORE` by itself.

- If a port should be marked `IGNORE` only on certain architectures, there are two other convenience variables that will automatically set `IGNORE` for you: `ONLY_FOR_ARCHS` and `NOT_FOR_ARCHS`. Examples:

```
ONLY_FOR_ARCHS= i386 amd64
```

```
NOT_FOR_ARCHS= ia64 sparc64
```

  A custom `IGNORE` message can be set using `ONLY_FOR_ARCHS_REASON` and `NOT_FOR_ARCHS_REASON`. Per architecture entries are possible with `ONLY_FOR_ARCHS_REASON_` *ARCH* and `NOT_FOR_ARCHS_REASON_` *ARCH*.

- If a port fetches i386 binaries and installs them, `IA32_BINARY_PORT` should be set. If this variable is set, it will be checked whether the `/usr/lib32` directory is available for IA32 versions of libraries and whether the kernel has IA32 compatibility compiled in. If one of these two dependencies is not satisfied, `IGNORE` will be set automatically.

12.13.2. Implementation Notes

The strings should not be quoted. Also, the wording of the string should be somewhat different due to the way the information is shown to the user. Examples:

```
BROKEN= fails to link with base --lcrypto
```

```
IGNORE= unsupported on recent versions
```

resulting in the following output from `make describe`:

```
===>  foobar-0.1 is marked as broken: fails to link with base --
lcrypto.
```

```
===>  foobar-0.1 is unsupported on recent versions.
```

12.14. Marking a Port for Removal with DEPRECATED or EXPIRATION_DATE

Do remember that BROKEN and FORBIDDEN are to be used as a temporary resort if a port is not working. Permanently broken ports should be removed from the tree entirely.

When it makes sense to do so, users can be warned about a pending port removal with DEPRECATED and EXPIRATION_DATE . The former is simply a string stating why the port is scheduled for removal; the latter is a string in ISO 8601 format (YYYY-MM-DD). Both will be shown to the user.

It is possible to set DEPRECATED without an EXPIRATION_DATE (for instance, recommending a newer version of the port), but the converse does not make any sense.

There is no set policy on how much notice to give. Current practice seems to be one month for security-related issues and two months for build issues. This also gives any interested committers a little time to fix the problems.

12.15. Avoid Use of the .error Construct

The correct way for a Makefile to signal that the port can not be installed due to some external factor (for instance, the user has specified an illegal combination of build options) is to set a non-blank value to IGNORE. This value will be formatted and shown to the user by make install.

It is a common mistake to use .error for this purpose. The problem with this is that many automated tools that work with the ports tree will fail in this situation. The most common occurrence of this is seen when trying to build /usr/ports/INDEX (see Section 9.1, "Running make describe"). However, even more trivial commands such as make maintainer also fail in this scenario. This is not acceptable.

Example 12.1. How to Avoid Using .error

The first of the next two Makefile snippets will cause make index to fail, while the second one will not:

```
.error -"option is not supported"
```

```
IGNORE=option is not supported
```

12.16. Usage of `sysctl`

The usage of `sysctl` is discouraged except in targets. This is because the evaluation of any `makevars`, such as used during `make index`, then has to run the command, further slowing down that process.

Usage of sysctl(8) should always be done with the `SYSCTL` variable, as it contains the fully qualified path and can be overridden, if one has such a special need.

12.17. Rerolling Distfiles

Sometimes the authors of software change the content of released distfiles without changing the file's name. You have to verify that the changes are official and have been performed by the author. It has happened in the past that the distfile was silently altered on the download servers with the intent to cause harm or compromise end user security.

Put the old distfile aside, download the new one, unpack them and compare the content with diff(1). If you see nothing suspicious, you can update `distinfo`. Be sure to summarize the differences in your PR or commit log, so that other people know that you have taken care to ensure that nothing bad has happened.

You might also want to contact the authors of the software and confirm the changes with them.

12.18. Avoiding Linuxisms

Do not use `/proc` if there are any other ways of getting the information, e.g., `setprogname(argv[0])` in `main()` and then getprogname(3) if you want to "know your name".

Do not rely on behaviour that is undocumented by POSIX.

Do not record timestamps in the critical path of the application if it also works without. Getting timestamps may be slow, depending on the accuracy of timestamps in the OS. If timestamps are really needed, determine how precise they have to be and use an API which is documented to just deliver the needed precision.

A number of simple syscalls (for example gettimeofday(2), getpid(2)) are much faster on Linux® than on any other operating system due to caching and the vsyscall performance

optimizations. Do not rely on them being cheap in performance-critical applications. In general, try hard to avoid syscalls if possible.

Do not rely on Linux®-specific socket behaviour. In particular, default socket buffer sizes are different (call setsockopt(2) with `SO_SNDBUF` and `SO_RCVBUF`, and while Linux®'s send(2) blocks when the socket buffer is full, FreeBSD's will fail and set `ENOBUFS` in errno.

If relying on non-standard behaviour is required, encapsulate it properly into a generic API, do a check for the behaviour in the configure stage, and stop if it is missing.

Check the man pages to see if the function used is a POSIX interface (in the "STANDARDS" section of the man page).

Do not assume that `/bin/sh` is bash. Ensure that a command line passed to system(3) will work with a POSIX compliant shell.

A list of common bashisms is available here.

Check that headers are included in the POSIX or man page recommended way, e.g., `sys/types.h` is often forgotten, which is not as much of a problem for Linux® as it is for FreeBSD.

Compile threaded applications with "-pthread", not "-lpthread" or variations thereof.

12.19. Miscellanea

The files `pkg-descr` and `pkg-plist` should each be double-checked. If you are reviewing a port and feel they can be worded better, do so.

Do not copy more copies of the GNU General Public License into our system, please.

Please be careful to note any legal issues! Do not let us illegally distribute software!

Chapter 13. A Sample `Makefile`

Here is a sample `Makefile` that you can use to create a new port. Make sure you remove all the extra comments (ones between brackets)!

It is recommended that you follow this format (ordering of variables, empty lines between sections, etc.). This format is designed so that the most important information is easy to locate. We recommend that you use portlint to check the `Makefile`.

```
[the header...just to make it easier for us to identify the ports.]
# Created by: Satoshi Asami <asami@FreeBSD.org>
[The optional Created by: line names the person who originally
created the port.  Note that the ":" is followed by a space
and not a tab character.
If this line is present, future maintainers should
not change or remove it except at the original author's request.]

# $FreeBSD$
[ ^^^^^^^^^ This will be automatically replaced with RCS ID string ↺
by SVN
when it is committed to our repository.  If upgrading a port, do ↺
not alter
this line back to -"$FreeBSD$".  SVN deals with it automatically.]

[section to describe the port itself and the master site -- PORTNAME
 and PORTVERSION are always first, followed by CATEGORIES,
 and then MASTER_SITES, which can be followed by MASTER_SITE_SUBDIR.
 PKGNAMEPREFIX and PKGNAMESUFFIX, if needed, will be after that.
 Then comes DISTNAME, EXTRACT_SUFX and/or DISTFILES, and then
 EXTRACT_ONLY, as necessary.]
PORTNAME= xdvi
PORTVERSION= 18.2
CATEGORIES= print
[do not forget the trailing slash ("/")!
 if you are not using MASTER_SITE_* macros]
MASTER_SITES= ${MASTER_SITE_XCONTRIB}
MASTER_SITE_SUBDIR= applications
PKGNAMEPREFIX= ja-
DISTNAME= xdvi-pl18
[set this if the source is not in the standard -".tar.gz" form]
EXTRACT_SUFX= .tar.Z

[section for distributed patches --- can be empty]
PATCH_SITES= ftp://ftp.sra.co.jp/pub/X11/japanese/
PATCHFILES= xdvi-18.patch1.gz xdvi-18.patch2.gz

[maintainer; *mandatory*!  This is the person who is volunteering to
 handle port updates, build breakages, and to whom a users can direct
```

```
 questions and bug reports.  To keep the quality of the Ports ↺
Collection
 as high as possible, we no longer accept new ports that are ↺
assigned to
 -"ports@FreeBSD.org".]
MAINTAINER= asami@FreeBSD.org
COMMENT= DVI Previewer for the X Window System

[dependencies --- can be empty]
RUN_DEPENDS= gs:${PORTSDIR}/print/ghostscript

[this section is for other standard bsd.port.mk variables that do not
 belong to any of the above]
[If it asks questions during configure, build, install...]
IS_INTERACTIVE= yes
[If it extracts to a directory other than ${DISTNAME}...]
WRKSRC=  ${WRKDIR}/xdvi-new
[If the distributed patches were not made relative to ${WRKSRC}, you
 may need to tweak this]
PATCH_DIST_STRIP= -p1
[If it requires a -"configure" script generated by GNU autoconf to ↺
be run]
GNU_CONFIGURE= yes
[If it requires GNU make, not -/usr/bin/make, to build...]
USES= gmake
[If it is an X application and requires -"xmkmf --a" to be run...]
USES= imake
[et cetera.]

[non-standard variables to be used in the rules below]
MY_FAVORITE_RESPONSE= "yeah, right"

[then the special rules, in the order they are called]
pre-fetch:
 i go fetch something, yeah

post-patch:
 i need to do something after patch, great

pre-install:
 and then some more stuff before installing, wow

[and then the epilogue]
.include <bsd.port.mk>
```

Chapter 14. Keeping Up

The FreeBSD Ports Collection is constantly changing. Here is some information on how to keep up.

14.1. FreshPorts

One of the easiest ways to learn about updates that have already been committed is by subscribing to FreshPorts. You can select multiple ports to monitor. Maintainers are strongly encouraged to subscribe, because they will receive notification of not only their own changes, but also any changes that any other FreeBSD committer has made. (These are often necessary to keep up with changes in the underlying ports framework—although it would be most polite to receive an advance heads-up from those committing such changes, sometimes this is overlooked or just simply impractical. Also, in some cases, the changes are very minor in nature. We expect everyone to use their best judgement in these cases.)

If you wish to use FreshPorts, all you need is an account. If your registered email address is `@FreeBSD.org` , you will see the opt-in link on the right hand side of the webpages. For those of you who already have a FreshPorts account, but are not using your `@FreeBSD.org` email address, just change your email to `@FreeBSD.org` , subscribe, then change it back again.

FreshPorts also has a sanity test feature which automatically tests each commit to the FreeBSD ports tree. If subscribed to this service, you will be notified of any errors which FreshPorts detects during sanity testing of your commits.

14.2. The Web Interface to the Source Repository

It is possible to browse the files in the source repository by using a web interface. Changes that affect the entire port system are now documented in the CHANGES file. Changes that affect individual ports are now documented in the UPDATING file. However, the definitive answer to any question is undoubtedly to read the source code of bsd.port.mk, and associated files.

14.3. The FreeBSD Ports Mailing List

If you maintain ports, you should consider following the FreeBSD ports mailing list. Important changes to the way ports work will be announced there, and then committed to `CHANGES` .

If this mailing list is too high volume you may consider following FreeBSD ports announce mailing list which is moderated and has no discussion.

14.4. The FreeBSD Port Building Cluster

One of the least-publicized strengths of FreeBSD is that an entire cluster of machines is dedicated to continually building the Ports Collection, for each of the major OS releases and for each Tier-1 architecture.

Individual ports are built unless they are specifically marked with `IGNORE`. Ports that are marked with `BROKEN` will still be attempted, to see if the underlying problem has been resolved. (This is done by passing `TRYBROKEN` to the port's `Makefile`.)

14.5. Portscout: the FreeBSD Ports Distfile Scanner

The build cluster is dedicated to building the latest release of each port with distfiles that have already been fetched. However, as the Internet continually changes, distfiles can quickly go missing. Portscout, the FreeBSD Ports distfile scanner, attempts to query every download site for every port to find out if each distfile is still available. Portscout can generate HTML reports and send emails about newly available ports to those who request them. Unless not otherwise subscribed, maintainers are asked to check periodically for changes, either by hand or using the RSS feed.

Portscout's first page gives the email address of the port maintainer, the number of ports the maintainer is responsible for, the number of those ports with new distfiles, and the percentage of those ports that are out-of-date. The search function allows for searching by email address for a specific maintainer, and for selecting whether or not only out-of-date ports should be shown.

Upon clicking on a maintainer's email address, a list of all of their ports is displayed, along with port category, current version number, whether or not there is a new version, when the port was last updated, and finally when it was last checked. A search function on this page allows the user to search for a specific port.

Clicking on a port name in the list displays the FreshPorts port information.

14.6. The FreeBSD Ports Monitoring System

Another handy resource is the FreeBSD Ports Monitoring System (also known as `portsmon`). This system comprises a database that processes information from several sources and allows it to be browsed via a web interface. Currently, the ports Problem Reports (PRs), the error logs from the build cluster, and individual files from the ports col-

lection are used. In the future, this will be expanded to include the distfile survey, as well as other sources.

To get started, you can view all information about a particular port by using the Overview of One Port.

As of this writing, this is the only resource available that maps GNATS PR entries to portnames. (PR submitters do not always include the portname in their Synopsis, although we would prefer that they did.) So, `portsmon` is a good place to start if you want to find out whether an existing port has any PRs filed against it and/or any build errors; or, to find out if a new port that you may be thinking about creating has already been submitted.

Chapter 15. Appendices

15.1. Values of USES

Table 15.1. Values of USES

Feature	Arguments	Description
ada	(none)	Depends on an Ada-capable compiler, and sets CC accordingly.
bison	(none), build, run, both	Implies that the port uses devel/bison in one way or another. By default, with no arguments or with the build argument, it implies bison as a build-time dependency, run implies a run-time dependency, and both implies both run-time and build-time dependencies.
charsetfix	(none)	Prevents the port from installing charset.alias . This should be installed only by converters/libiconv. CHARSETFIX_MAKEFILEIN can be set to a path relative to WRKSRC if charset.alias is not installed by WRKSRC / Makefile.in .
cmake	(none), outsource	The port will use CMake for configuring and building. With the outsource argument, an out-of-source build will be performed. For more information see Section 6.4.4, "Using cmake".
compiler	(none), c++11-lang , c++11-lib, c11, openmp, features	Determines which compiler to use based on any given wishes. Use c++11-lang if the port needs a C+11-ca-

Feature	Arguments	Description
		pable compiler, and `c++11-lib` if the port also needs a C++11-ready standard library. If the port needs a compiler understanding C11 or OpenMP, the corresponding parameters can be used. Use `features` to request a list of features supported by the default compiler. After including `bsd.port.pre.mk` the port can inspect the results using these variables: • `COMPILER_TYPE` : the default compiler on the system, either gcc or clang • `ALT_COMPILER_TYPE` : the alternative compiler on the system, either gcc or clang. Only set if two compilers are present in the base system. • `COMPILER_VERSION` : the first two digits of the version of the default compiler. • `ALT_COMPILER_VERSION` : the first two digits of the version of the alternative compiler, if present. • `CHOSEN_COMPILER_TYPE` : the chosen compiler, either gcc or clang • `COMPILER_FEATURES` : the features supported by the default compiler. It currently lists the C++ library.

Feature	Arguments	Description
`desktop-file-utils`	(none)	Implies that the port uses the update-desktop-database from `devel/desktop-file-utils`. An extra post-install step will be run without interfering with any post-install steps already in the port `Makefile`. Lines will be inserted into the plist to run update-desktop-database on package install or removal.
`display`	(none), `ARGS`	Set up a virtual display environment. If the environment variable `DISPLAY` is not set, then Xvfb is added as a build dependency, and `CONFIGURE_ENV` is extended with the port number of the currently running instance of Xvfb. The `ARGS` parameter defaults to `install` and controls the phase around which to start and stop the virtual display.
`dos2unix`	(none)	The port has files with line endings in DOS format which need to be converted. Three variables can be set to control which files will be converted. The default is to convert *all* files, including binaries. • `DOS2UNIX_REGEX`: match file names based on a regular expression. • `DOS2UNIX_FILES`: match literal file names.

Feature	Arguments	Description
		• DOS2UNIX_GLOB : match file names based on a glob pattern.
fmake	(none)	Implies that the port uses devel/fmake as build-time dependency.
fuse	(none)	Implies the port will depend on the FUSE library and handle the dependency on the kernel module depending on the version of FreeBSD.
gettext	(none), lib, build, run	Implies that the port uses devel/gettext in one way or another. By default, with no arguments or with the lib argument, implies gettext with build-time and run-time dependencies, build implies a build-time dependency, and run implies a run-time dependency.
gmake	(none)	Implies that the port uses devel/gmake as build-time dependency.
iconv	(none), lib, build, patch	Implies that the port uses iconv functions, either from the port converters/libiconv as a build-time and run-time dependency, or from the base system on 10-CURRENT after a native iconv was committed in <revnumber>254273</revnumber>. By default, with no arguments or with the lib argument, implies iconv with build-time and run-time dependencies, build implies a

Feature	Arguments	Description
		build-time dependency, and `patch` implies a patch-time dependency. For more information see Section 6.21, "Using `iconv`".
`imake`	(none)	Implies that the port uses `devel/imake` as build-time dependency.
`kmod`	(none)	Fills in the boilerplate for kernel module ports, currently: • Add `kld` to `CATEGORIES`. • Set `SSP_UNSAFE` . • Set `IGNORE` if the kernel sources are not found in `SRC_BASE` . • Define `KMODDIR` to `/boot/modules` by default, add it to `PLIST_SUB` and `MAKE_ENV` , and create it upon installation. If `KMODDIR` is set to `/boot/kernel`, it will be rewritten to `/boot/modules` . This prevents breaking packages when upgrading the kernel due to `/boot/kernel` being renamed to `/boot/kernel.old` in the process. • Handle cross-referencing kernel modules upon installation and deinstallation.
`motif`	(none)	Implies that the ports uses `devel/open-motif` as a library dependency. End users can set `WANT_LESSTIF`

Feature	Arguments	Description
		for the dependency to be on `devel/lesstif` instead of `devel/open-motif`.
`ncurses`	(none), `base`, `port`	Implies that the port uses ncurses, and causes some useful variables to be set.
`pathfix`	(none)	Look for the `Makefile.in` and `configure` files in the port's associated sources and fix common paths to make sure they respect the FreeBSD hierarchy.
`perl5`	(none)	Implies a dependency on Perl. The following variables can be set: • `PERL_VERSION`: Full version of Perl to use, or the default if not set • `PERL_ARCH`: Directory name of architecture dependent libraries, defaults to `mach` • `PERL_PORT`: Name of the Perl port to be installed, the default is derived from `PERL_VERSION` • `SITE_PERL`: Directory name for site specific Perl packages • `USE_PERL5`: Phases in which to use Perl, can be `extract`, `patch`, `build`, `install`, or `run`. It can also be `configure`, `modbuild`, or `modbuildtiny` when `Makefile.PL`, `Build.PL`, or the Module::Build::Tiny

Feature	Arguments	Description
		flavor of `Build.PL` is required. Defaults to `build run`.
`pkgconfig`	(none), `build`, `run`, `both`	Implies that the port uses `devel/pkgconf` in one way or another. With no arguments or with the `build` argument, it implies `pkg-config` as a build-time dependency; `run` implies a run-time dependency; and `both` implies both run-time and build-time dependencies.
`pure`	(none), `ffi`	Implies that the port uses `lang/pure` in one way or another, but largely is used for building related pure ports. With the `ffi` argument, it implies `devel/pure-ffi` as a run-time dependency.
`qmail`	(none), `build`, `run`, `both`, `vars`	Implies that the port uses `mail/qmail` in one way or another. With the `build` argument, it implies `qmail` as a build-time dependency. `run` implies a run-time dependency. Using no argument or the `both` argument implies both run-time and build-time dependencies. `vars` will only set QMAIL variables for the port to use.
`qmake`	(none), `norecursive`	The port will use QMake for configuring. For more information see Section 6.10.3, “Using `qmake`”.
`readline`	(none), port	Implies that the port uses readline as library dependency, and sets `CPPFLAGS` and `LDFLAGS` as necessary.

Feature	Arguments	Description
shared-mime-info	(none)	Implies that the port uses update-mime-database from misc/shared-mime-info. This uses will automatically add a post-install step in such a way that the port itself still can specify there own post-install step if needed. It also insert lines into the plist for package install and removal to run update-mime-data with the correct arguments.
shebangfix	(none)	A lot of software uses incorrect locations for script interpreters, most notably /usr/bin/perl and /bin/bash. This fixes shebang lines in scripts listed in SHEBANG_FILES . Currently Perl, Python, Bash, Ruby, and PHP are supported by default. To support another interpreter, set SHEBANG_LANG (for example SHEBANG_LANG=lua), then lua_OLD_CMD and lua_CMD .
uidfix	(none)	Changes some default behavior (mostly variables) of the build system to allow installing this port as a normal user. Try this in your port before adding NEED_ROOT=yes
webplugin	(none), ARGS	Automatically create and remove symbolic links for each application that supports the webplugin framework. ARGS can be one of: • gecko: support plug-ins based on Gecko

Feature	Arguments	Description
		• `native`: support plug-ins for Gecko, Opera, and WebKit-GTK • `linux`: support Linux plug-ins • `all` (default, implicit): support all plug-in types • (individual entries): support only the browsers listed The following variables can be adjusted: • `WEBPLUGIN_FILES` : No default, must be set manually. The plug-in files to install. • `WEBPLUGIN_DIR` : The directory to install the plug-in files to, default *`PREFIX`*`/lib/browser_plugins/`*`WEBPLUGIN_NAME`*`.` Set this if the port installs plug-in files outside of the default directory to prevent broken symbolic links. • `WEBPLUGIN_NAME` : The final directory to install the plug-in files into, default `PKGBASE`.
`zenoss`	(none)	Implies the port uses `net-mgmt/zenoss` in one way or another, but largely is used for building zenoss related zenpack ports.

Feature	Arguments	Description
zope	(none)	Implies that the port uses www/zope in one way or another, but is largely used for building zope related ports. ZOPE_VERSION can be used by a port to indicate that a specific version of zope shall be used.

15.2. __FreeBSD_version Values

Here is a convenient list of __FreeBSD_version values as defined in sys/param.h:

Table 15.2. __FreeBSD_version Values

Value	Date	Release
119411		2.0-RELEASE
199501, 199503	March 19, 1995	2.1-CURRENT
199504	April 9, 1995	2.0.5-RELEASE
199508	August 26, 1995	2.2-CURRENT before 2.1
199511	November 10, 1995	2.1.0-RELEASE
199512	November 10, 1995	2.2-CURRENT before 2.1.5
199607	July 10, 1996	2.1.5-RELEASE
199608	July 12, 1996	2.2-CURRENT before 2.1.6
199612	November 15, 1996	2.1.6-RELEASE
199612		2.1.7-RELEASE
220000	February 19, 1997	2.2-RELEASE
(not changed)		2.2.1-RELEASE
(not changed)		2.2-STABLE after 2.2.1-RELEASE
221001	April 15, 1997	2.2-STABLE after texinfo-3.9
221002	April 30, 1997	2.2-STABLE after top
222000	May 16, 1997	2.2.2-RELEASE
222001	May 19, 1997	2.2-STABLE after 2.2.2-RELEASE

Value	Date	Release
225000	October 2, 1997	2.2.5-RELEASE
225001	November 20, 1997	2.2-STABLE after 2.2.5-RELEASE
225002	December 27, 1997	2.2-STABLE after ldconfig -R merge
226000	March 24, 1998	2.2.6-RELEASE
227000	July 21, 1998	2.2.7-RELEASE
227001	July 21, 1998	2.2-STABLE after 2.2.7-RELEASE
227002	September 19, 1998	2.2-STABLE after semctl(2) change
228000	November 29, 1998	2.2.8-RELEASE
228001	November 29, 1998	2.2-STABLE after 2.2.8-RELEASE
300000	February 19, 1996	3.0-CURRENT before mount(2) change
300001	September 24, 1997	3.0-CURRENT after mount(2) change
300002	June 2, 1998	3.0-CURRENT after semctl(2) change
300003	June 7, 1998	3.0-CURRENT after ioctl arg changes
300004	September 3, 1998	3.0-CURRENT after ELF conversion
300005	October 16, 1998	3.0-RELEASE
300006	October 16, 1998	3.0-CURRENT after 3.0-RELEASE
300007	January 22, 1999	3.0-STABLE after 3/4 branch
310000	February 9, 1999	3.1-RELEASE
310001	March 27, 1999	3.1-STABLE after 3.1-RELEASE
310002	April 14, 1999	3.1-STABLE after C++ constructor/destructor order change
320000		3.2-RELEASE

Value	Date	Release
320001	May 8, 1999	3.2-STABLE
320002	August 29, 1999	3.2-STABLE after binary-incompatible IPFW and socket changes
330000	September 2, 1999	3.3-RELEASE
330001	September 16, 1999	3.3-STABLE
330002	November 24, 1999	3.3-STABLE after adding mkstemp(3) to libc
340000	December 5, 1999	3.4-RELEASE
340001	December 17, 1999	3.4-STABLE
350000	June 20, 2000	3.5-RELEASE
350001	July 12, 2000	3.5-STABLE
400000	January 22, 1999	4.0-CURRENT after 3.4 branch
400001	February 20, 1999	4.0-CURRENT after change in dynamic linker handling
400002	March 13, 1999	4.0-CURRENT after C++ constructor/destructor order change
400003	March 27, 1999	4.0-CURRENT after functioning dladdr(3)
400004	April 5, 1999	4.0-CURRENT after __deregister_frame_info dynamic linker bug fix (also 4.0-CURRENT after EGCS 1.1.2 integration)
400005	April 27, 1999	4.0-CURRENT after suser(9) API change (also 4.0-CURRENT after newbus)
400006	May 31, 1999	4.0-CURRENT after cdevsw registration change
400007	June 17, 1999	4.0-CURRENT after the addition of so_cred for socket level credentials

Value	Date	Release
400008	June 20, 1999	4.0-CURRENT after the addition of a poll syscall wrapper to libc_r
400009	July 20, 1999	4.0-CURRENT after the change of the kernel's `dev_t` type to `struct specinfo` pointer
400010	September 25, 1999	4.0-CURRENT after fixing a hole in jail(2)
400011	September 29, 1999	4.0-CURRENT after the `sigset_t` datatype change
400012	November 15, 1999	4.0-CURRENT after the cutover to the GCC 2.95.2 compiler
400013	December 4, 1999	4.0-CURRENT after adding pluggable linux-mode ioctl handlers
400014	January 18, 2000	4.0-CURRENT after importing OpenSSL
400015	January 27, 2000	4.0-CURRENT after the C++ ABI change in GCC 2.95.2 from -fvtable-thunks to -fno-vtable-thunks by default
400016	February 27, 2000	4.0-CURRENT after importing OpenSSH
400017	March 13, 2000	4.0-RELEASE
400018	March 17, 2000	4.0-STABLE after 4.0-RELEASE
400019	May 5, 2000	4.0-STABLE after the introduction of delayed checksums.
400020	June 4, 2000	4.0-STABLE after merging libxpg4 code into libc.
400021	July 8, 2000	4.0-STABLE after upgrading Binutils to 2.10.0, ELF brand-

Value	Date	Release
		ing changes, and tcsh in the base system.
410000	July 14, 2000	4.1-RELEASE
410001	July 29, 2000	4.1-STABLE after 4.1-RELEASE
410002	September 16, 2000	4.1-STABLE after setproctitle(3) moved from libutil to libc.
411000	September 25, 2000	4.1.1-RELEASE
411001		4.1.1-STABLE after 4.1.1-RELEASE
420000	October 31, 2000	4.2-RELEASE
420001	January 10, 2001	4.2-STABLE after combining libgcc.a and libgcc_r.a, and associated GCC linkage changes.
430000	March 6, 2001	4.3-RELEASE
430001	May 18, 2001	4.3-STABLE after wint_t introduction.
430002	July 22, 2001	4.3-STABLE after PCI powerstate API merge.
440000	August 1, 2001	4.4-RELEASE
440001	October 23, 2001	4.4-STABLE after d_thread_t introduction.
440002	November 4, 2001	4.4-STABLE after mount structure changes (affects filesystem klds).
440003	December 18, 2001	4.4-STABLE after the userland components of smbfs were imported.
450000	December 20, 2001	4.5-RELEASE
450001	February 24, 2002	4.5-STABLE after the usb structure element rename.
450004	April 16, 2002	4.5-STABLE after the `sendmail_enable`

Value	Date	Release
		rc.conf(5) variable was made to take the value `NONE`.
450005	April 27, 2002	4.5-STABLE after moving to XFree86 4 by default for package builds.
450006	May 1, 2002	4.5-STABLE after accept filtering was fixed so that is no longer susceptible to an easy DoS.
460000	June 21, 2002	4.6-RELEASE
460001	June 21, 2002	4.6-STABLE sendfile(2) fixed to comply with documentation, not to count any headers sent against the amount of data to be sent from the file.
460002	July 19, 2002	4.6.2-RELEASE
460100	June 26, 2002	4.6-STABLE
460101	June 26, 2002	4.6-STABLE after MFC of `sed -i'.
460102	September 1, 2002	4.6-STABLE after MFC of many new pkg_install features from the HEAD.
470000	October 8, 2002	4.7-RELEASE
470100	October 9, 2002	4.7-STABLE
470101	November 10, 2002	Start generated __std{in,out,err}p references rather than __sF. This changes std{in,out,err} from a compile time expression to a runtime one.
470102	January 23, 2003	4.7-STABLE after MFC of mbuf changes to replace m_aux mbufs by m_tag's
470103	February 14, 2003	4.7-STABLE gets OpenSSL 0.9.7
480000	March 30, 2003	4.8-RELEASE

Value	Date	Release
480100	April 5, 2003	4.8-STABLE
480101	May 22, 2003	4.8-STABLE after realpath(3) has been made thread-safe
480102	August 10, 2003	4.8-STABLE 3ware API changes to twe.
490000	October 27, 2003	4.9-RELEASE
490100	October 27, 2003	4.9-STABLE
490101	January 8, 2004	4.9-STABLE after e_sid was added to struct kinfo_eproc.
490102	February 4, 2004	4.9-STABLE after MFC of libmap functionality for rtld.
491000	May 25, 2004	4.10-RELEASE
491100	June 1, 2004	4.10-STABLE
491101	August 11, 2004	4.10-STABLE after MFC of revision 20040629 of the package tools
491102	November 16, 2004	4.10-STABLE after VM fix dealing with unwiring of fictitious pages
492000	December 17, 2004	4.11-RELEASE
492100	December 17, 2004	4.11-STABLE
492101	April 18, 2006	4.11-STABLE after adding libdata/ldconfig directories to mtree files.
500000	March 13, 2000	5.0-CURRENT
500001	April 18, 2000	5.0-CURRENT after adding addition ELF header fields, and changing our ELF binary branding method.
500002	May 2, 2000	5.0-CURRENT after kld metadata changes.
500003	May 18, 2000	5.0-CURRENT after buf/bio changes.
500004	May 26, 2000	5.0-CURRENT after binutils upgrade.

Value	Date	Release
500005	June 3, 2000	5.0-CURRENT after merging libxpg4 code into libc and after TASKQ interface introduction.
500006	June 10, 2000	5.0-CURRENT after the addition of AGP interfaces.
500007	June 29, 2000	5.0-CURRENT after Perl upgrade to 5.6.0
500008	July 7, 2000	5.0-CURRENT after the update of KAME code to 2000/07 sources.
500009	July 14, 2000	5.0-CURRENT after ether_ifattach() and ether_ifdetach() changes.
500010	July 16, 2000	5.0-CURRENT after changing mtree defaults back to original variant, adding -L to follow symlinks.
500011	July 18, 2000	5.0-CURRENT after kqueue API changed.
500012	September 2, 2000	5.0-CURRENT after setproctitle(3) moved from libutil to libc.
500013	September 10, 2000	5.0-CURRENT after the first SMPng commit.
500014	January 4, 2001	5.0-CURRENT after <sys/select.h> moved to <sys/selinfo.h>.
500015	January 10, 2001	5.0-CURRENT after combining libgcc.a and libgcc_r.a, and associated GCC linkage changes.
500016	January 24, 2001	5.0-CURRENT after change allowing libc and libc_r to be linked together, deprecating -pthread option.
500017	February 18, 2001	5.0-CURRENT after switch from struct ucred to

Value	Date	Release
		struct xucred to stabilize kernel-exported API for mountd et al.
500018	February 24, 2001	5.0-CURRENT after addition of CPUTYPE make variable for controlling CPU-specific optimizations.
500019	June 9, 2001	5.0-CURRENT after moving machine/ioctl_fd.h to sys/fdcio.h
500020	June 15, 2001	5.0-CURRENT after locale names renaming.
500021	June 22, 2001	5.0-CURRENT after Bzip2 import. Also signifies removal of S/Key.
500022	July 12, 2001	5.0-CURRENT after SSE support.
500023	September 14, 2001	5.0-CURRENT after KSE Milestone 2.
500024	October 1, 2001	5.0-CURRENT after d_thread_t, and moving UUCP to ports.
500025	October 4, 2001	5.0-CURRENT after ABI change for descriptor and creds passing on 64 bit platforms.
500026	October 9, 2001	5.0-CURRENT after moving to XFree86 4 by default for package builds, and after the new libc strnstr() function was added.
500027	October 10, 2001	5.0-CURRENT after the new libc strcasestr() function was added.
500028	December 14, 2001	5.0-CURRENT after the userland components of smbfs were imported.

Value	Date	Release
(not changed)		5.0-CURRENT after the new C99 specific-width integer types were added.
500029	January 29, 2002	5.0-CURRENT after a change was made in the return value of sendfile(2).
500030	February 15, 2002	5.0-CURRENT after the introduction of the type `fflags_t`, which is the appropriate size for file flags.
500031	February 24, 2002	5.0-CURRENT after the usb structure element rename.
500032	March 16, 2002	5.0-CURRENT after the introduction of Perl 5.6.1.
500033	April 3, 2002	5.0-CURRENT after the `sendmail_enable` rc.conf(5) variable was made to take the value `NONE`.
500034	April 30, 2002	5.0-CURRENT after mtx_init() grew a third argument.
500035	May 13, 2002	5.0-CURRENT with Gcc 3.1.
500036	May 17, 2002	5.0-CURRENT without Perl in /usr/src
500037	May 29, 2002	5.0-CURRENT after the addition of dlfunc(3)
500038	July 24, 2002	5.0-CURRENT after the types of some struct sockbuf members were changed and the structure was reordered.
500039	September 1, 2002	5.0-CURRENT after GCC 3.2.1 import. Also after headers stopped using _BSD_FOO_T_ and started using _FOO_T_DECLARED. This value can also be used as a conservative estimate of

Value	Date	Release
		the start of bzip2(1) package support.
500040	September 20, 2002	5.0-CURRENT after various changes to disk functions were made in the name of removing dependency on disklabel structure internals.
500041	October 1, 2002	5.0-CURRENT after the addition of getopt_long(3) to libc.
500042	October 15, 2002	5.0-CURRENT after Binutils 2.13 upgrade, which included new FreeBSD emulation, vec, and output format.
500043	November 1, 2002	5.0-CURRENT after adding weak pthread_XXX stubs to libc, obsoleting libXThrStub.so. 5.0-RELEASE.
500100	January 17, 2003	5.0-CURRENT after branching for RELENG_5_0
500101	February 19, 2003	<sys/dkstat.h> is empty and should not be included.
500102	February 25, 2003	5.0-CURRENT after the d_mmap_t interface change.
500103	February 26, 2003	5.0-CURRENT after taskqueue_swi changed to run without Giant, and taskqueue_swi_giant added to run with Giant.
500104	February 27, 2003	cdevsw_add() and cdevsw_remove() no longer exists. Appearance of MAJOR_AUTO allocation facility.
500105	March 4, 2003	5.0-CURRENT after new cdevsw initialization method.

Value	Date	Release
500106	March 8, 2003	devstat_add_entry() has been replaced by devstat_new_entry()
500107	March 15, 2003	Devstat interface change; see sys/sys/param.h 1.149
500108	March 15, 2003	Token-Ring interface changes.
500109	March 25, 2003	Addition of vm_paddr_t.
500110	March 28, 2003	5.0-CURRENT after realpath(3) has been made thread-safe
500111	April 9, 2003	5.0-CURRENT after usbhid(3) has been synced with NetBSD
500112	April 17, 2003	5.0-CURRENT after new NSS implementation and addition of POSIX.1 getpw*_r, getgr*_r functions
500113	May 2, 2003	5.0-CURRENT after removal of the old rc system.
501000	June 4, 2003	5.1-RELEASE.
501100	June 2, 2003	5.1-CURRENT after branching for RELENG_5_1.
501101	June 29, 2003	5.1-CURRENT after correcting the semantics of sigtimedwait(2) and sigwaitinfo(2).
501102	July 3, 2003	5.1-CURRENT after adding the lockfunc and lockfuncarg fields to bus_dma_tag_create(9).
501103	July 31, 2003	5.1-CURRENT after GCC 3.3.1-pre 20030711 snapshot integration.
501104	August 5, 2003	5.1-CURRENT 3ware API changes to twe.

Value	Date	Release
501105	August 17, 2003	5.1-CURRENT dynamically-linked /bin and /sbin support and movement of libraries to /lib.
501106	September 8, 2003	5.1-CURRENT after adding kernel support for Coda 6.x.
501107	September 17, 2003	5.1-CURRENT after 16550 UART constants moved from <dev/sio/sioreg.h> to <dev/ic/ns16550.h> . Also when libmap functionality was unconditionally supported by rtld.
501108	September 23, 2003	5.1-CURRENT after PFIL_HOOKS API update
501109	September 27, 2003	5.1-CURRENT after adding kiconv(3)
501110	September 28, 2003	5.1-CURRENT after changing default operations for open and close in cdevsw
501111	October 16, 2003	5.1-CURRENT after changed layout of cdevsw
501112	October 16, 2003	5.1-CURRENT after adding kobj multiple inheritance
501113	October 31, 2003	5.1-CURRENT after the if_xname change in struct ifnet
501114	November 16, 2003	5.1-CURRENT after changing /bin and /sbin to be dynamically linked
502000	December 7, 2003	5.2-RELEASE
502010	February 23, 2004	5.2.1-RELEASE
502100	December 7, 2003	5.2-CURRENT after branching for RELENG_5_2
502101	December 19, 2003	5.2-CURRENT after __cxa_atexit/__cxa_finalize functions were added to libc.

Value	Date	Release
502102	January 30, 2004	5.2-CURRENT after change of default thread library from libc_r to libpthread.
502103	February 21, 2004	5.2-CURRENT after device driver API megapatch.
502104	February 25, 2004	5.2-CURRENT after getopt_long_only() addition.
502105	March 5, 2004	5.2-CURRENT after NULL is made into ((void *)0) for C, creating more warnings.
502106	March 8, 2004	5.2-CURRENT after pf is linked to the build and install.
502107	March 10, 2004	5.2-CURRENT after time_t is changed to a 64-bit value on sparc64.
502108	March 12, 2004	5.2-CURRENT after Intel C/C++ compiler support in some headers and execve(2) changes to be more strictly conforming to POSIX.
502109	March 22, 2004	5.2-CURRENT after the introduction of the bus_alloc_resource_any API
502110	March 27, 2004	5.2-CURRENT after the addition of UTF-8 locales
502111	April 11, 2004	5.2-CURRENT after the removal of the getvfsent(3) API
502112	April 13, 2004	5.2-CURRENT after the addition of the .warning directive for make.
502113	June 4, 2004	5.2-CURRENT after ttyioctl() was made mandatory for serial drivers.
502114	June 13, 2004	5.2-CURRENT after import of the ALTQ framework.

Value	Date	Release
502115	June 14, 2004	5.2-CURRENT after changing sema_timedwait(9) to return 0 on success and a non-zero error code on failure.
502116	June 16, 2004	5.2-CURRENT after changing kernel dev_t to be pointer to struct cdev *.
502117	June 17, 2004	5.2-CURRENT after changing kernel udev_t to dev_t.
502118	June 17, 2004	5.2-CURRENT after adding support for CLOCK_VIRTUAL and CLOCK_PROF to clock_gettime(2) and clock_getres(2).
502119	June 22, 2004	5.2-CURRENT after changing network interface cloning overhaul.
502120	July 2, 2004	5.2-CURRENT after the update of the package tools to revision 20040629.
502121	July 9, 2004	5.2-CURRENT after marking Bluetooth code as non-i386 specific.
502122	July 11, 2004	5.2-CURRENT after the introduction of the KDB debugger framework, the conversion of DDB into a backend and the introduction of the GDB backend.
502123	July 12, 2004	5.2-CURRENT after change to make VFS_ROOT take a struct thread argument as does vflush. Struct kinfo_proc now has a user data pointer. The switch of the default X implementation to `xorg` was also made at this time.

Value	Date	Release
502124	July 24, 2004	5.2-CURRENT after the change to separate the way ports rc.d and legacy scripts are started.
502125	July 28, 2004	5.2-CURRENT after the back-out of the previous change.
502126	July 31, 2004	5.2-CURRENT after the removal of kmem_alloc_pageable() and the import of gcc 3.4.2.
502127	August 2, 2004	5.2-CURRENT after changing the UMA kernel API to allow ctors/inits to fail.
502128	August 8, 2004	5.2-CURRENT after the change of the vfs_mount signature as well as global replacement of PRISON_ROOT with SUSER_ALLOWJAIL for the suser(9) API.
503000	August 23, 2004	5.3-BETA/RC before the pfil API change
503001	September 22, 2004	5.3-RELEASE
503100	October 16, 2004	5.3-STABLE after branching for RELENG_5_3
503101	December 3, 2004	5.3-STABLE after addition of glibc style strftime(3) padding options.
503102	February 13, 2005	5.3-STABLE after OpenBSD's nc(1) import MFC.
503103	February 27, 2005	5.4-PRERELEASE after the MFC of the fixes in `<src/include/stdbool.h>` and `<src/sys/i386/include/_types.h>` for using the GCC-compatibility of the Intel C/C++ compiler.
503104	February 28, 2005	5.4-PRERELEASE after the MFC of the change of

Value	Date	Release
		ifi_epoch from wall clock time to uptime.
503105	March 2, 2005	5.4-PRERELEASE after the MFC of the fix of EOVERFLOW check in vswprintf(3).
504000	April 3, 2005	5.4-RELEASE.
504100	April 3, 2005	5.4-STABLE after branching for RELENG_5_4
504101	May 11, 2005	5.4-STABLE after increasing the default thread stacksizes
504102	June 24, 2005	5.4-STABLE after the addition of sha256
504103	October 3, 2005	5.4-STABLE after the MFC of if_bridge
504104	November 13, 2005	5.4-STABLE after the MFC of bsdiff and portsnap
504105	January 17, 2006	5.4-STABLE after MFC of ldconfig_local_dirs change.
505000	May 12, 2006	5.5-RELEASE.
505100	May 12, 2006	5.5-STABLE after branching for RELENG_5_5
600000	August 18, 2004	6.0-CURRENT
600001	August 27, 2004	6.0-CURRENT after permanently enabling PFIL_HOOKS in the kernel.
600002	August 30, 2004	6.0-CURRENT after initial addition of ifi_epoch to struct if_data. Backed out after a few days. Do not use this value.
600003	September 8, 2004	6.0-CURRENT after the re-addition of the ifi_epoch member of struct if_data.
600004	September 29, 2004	6.0-CURRENT after addition of the struct inpcb argument to the pfil API.

Value	Date	Release
600005	October 5, 2004	6.0-CURRENT after addition of the "-d DESTDIR" argument to newsyslog.
600006	November 4, 2004	6.0-CURRENT after addition of glibc style strftime(3) padding options.
600007	December 12, 2004	6.0-CURRENT after addition of 802.11 framework updates.
600008	January 25, 2005	6.0-CURRENT after changes to VOP_*VOBJECT() functions and introduction of MNTK_MPSAFE flag for Giantfree filesystems.
600009	February 4, 2005	6.0-CURRENT after addition of the cpufreq framework and drivers.
600010	February 6, 2005	6.0-CURRENT after importing OpenBSD's nc(1).
600011	February 12, 2005	6.0-CURRENT after removing semblance of SVID2 `matherr()` support.
600012	February 15, 2005	6.0-CURRENT after increase of default thread stacks' size.
600013	February 19, 2005	6.0-CURRENT after fixes in `<src/include/stdbool.h>` and `<src/sys/i386/include/_types.h>` for using the GCC-compatibility of the Intel C/C++ compiler.
600014	February 21, 2005	6.0-CURRENT after EOVERFLOW checks in vswprintf(3) fixed.
600015	February 25, 2005	6.0-CURRENT after changing the struct if_data member, ifi_epoch, from wall clock time to uptime.

Value	Date	Release
600016	February 26, 2005	6.0-CURRENT after LC_CTYPE disk format changed.
600017	February 27, 2005	6.0-CURRENT after NLS catalogs disk format changed.
600018	February 27, 2005	6.0-CURRENT after LC_COLLATE disk format changed.
600019	February 28, 2005	Installation of acpica includes into /usr/include.
600020	March 9, 2005	Addition of MSG_NOSIGNAL flag to send(2) API.
600021	March 17, 2005	Addition of fields to cdevsw
600022	March 21, 2005	Removed gtar from base system.
600023	April 13, 2005	LOCAL_CREDS, LOCAL_CONNWAIT socket options added to unix(4).
600024	April 19, 2005	hwpmc(4) and related tools added to 6.0-CURRENT.
600025	April 26, 2005	struct icmphdr added to 6.0-CURRENT.
600026	May 3, 2005	pf updated to 3.7.
600027	May 6, 2005	Kernel libalias and ng_nat introduced.
600028	May 13, 2005	POSIX ttyname_r(3) made available through unistd.h and libc.
600029	May 29, 2005	6.0-CURRENT after libpcap updated to v0.9.1 alpha 096.
600030	June 5, 2005	6.0-CURRENT after importing NetBSD's if_bridge(4).
600031	June 10, 2005	6.0-CURRENT after struct ifnet was broken out of the driver softcs.

Value	Date	Release
600032	July 11, 2005	6.0-CURRENT after the import of libpcap v0.9.1.
600033	July 25, 2005	6.0-STABLE after bump of all shared library versions that had not been changed since RELENG_5.
600034	August 13, 2005	6.0-STABLE after credential argument is added to dev_clone event handler. 6.0-RELEASE.
600100	November 1, 2005	6.0-STABLE after 6.0-RELEASE
600101	December 21, 2005	6.0-STABLE after incorporating scripts from the local_startup directories into the base rcorder(8).
600102	December 30, 2005	6.0-STABLE after updating the ELF types and constants.
600103	January 15, 2006	6.0-STABLE after MFC of pidfile(3) API.
600104	January 17, 2006	6.0-STABLE after MFC of ldconfig_local_dirs change.
600105	February 26, 2006	6.0-STABLE after NLS catalog support of csh(1).
601000	May 6, 2006	6.1-RELEASE
601100	May 6, 2006	6.1-STABLE after 6.1-RELEASE.
601101	June 22, 2006	6.1-STABLE after the import of csup.
601102	July 11, 2006	6.1-STABLE after the iwi(4) update.
601103	July 17, 2006	6.1-STABLE after the resolver update to BIND9, and exposure of reentrant version of netdb functions.
601104	August 8, 2006	6.1-STABLE after DSO (dynamic shared objects) sup-

Value	Date	Release
		port has been enabled in OpenSSL.
601105	September 2, 2006	6.1-STABLE after 802.11 fix-ups changed the api for the IEEE80211_IOC_STA_INFO ioctl.
602000	November 15, 2006	6.2-RELEASE
602100	September 15, 2006	6.2-STABLE after 6.2-RELEASE.
602101	December 12, 2006	6.2-STABLE after the addition of Wi-Spy quirk.
602102	December 28, 2006	6.2-STABLE after pci_find_extcap() addition.
602103	January 16, 2007	6.2-STABLE after MFC of dlsym change to look for a requested symbol both in specified dso and its implicit dependencies.
602104	January 28, 2007	6.2-STABLE after MFC of ng_deflate(4) and ng_pred1(4) netgraph nodes and new compression and encryption modes for ng_ppp(4) node.
602105	February 20, 2007	6.2-STABLE after MFC of BSD licensed version of gzip(1) ported from NetBSD.
602106	March 31, 2007	6.2-STABLE after MFC of PCI MSI and MSI-X support.
602107	April 6, 2007	6.2-STABLE after MFC of ncurses 5.6 and wide character support.
602108	April 11, 2007	6.2-STABLE after MFC of CAM 'SG' peripheral device, which implements a subset of Linux SCSI SG passthrough device API.

Value	Date	Release
602109	April 17, 2007	6.2-STABLE after MFC of readline 5.2 patchset 002.
602110	May 2, 2007	6.2-STABLE after MFC of pmap_invalidate_cache(), pmap_change_attr(), pmap_mapbios(), pmap_mapdev_attr(), and pmap_unmapbios() for amd64 and i386.
602111	June 11, 2007	6.2-STABLE after MFC of BOP_BDFLUSH and caused breakage of the filesystem modules KBI.
602112	September 21, 2007	6.2-STABLE after libutil(3) MFC's.
602113	October 25, 2007	6.2-STABLE after MFC of wide and single byte ctype separation. Newly compiled binary that references to ctype.h may require a new symbol, __mb_sb_limit, which is not available on older systems.
602114	October 30, 2007	6.2-STABLE after ctype ABI forward compatibility restored.
602115	November 21, 2007	6.2-STABLE after back out of wide and single byte ctype separation.
603000	November 25, 2007	6.3-RELEASE
603100	November 25, 2007	6.3-STABLE after 6.3-RELEASE.
603101	December 7, 2007	6.3-STABLE after fixing multibyte type support in bit macro.
603102	April 24, 2008	6.3-STABLE after adding l_sysid to struct flock.

Value	Date	Release
603103	May 27, 2008	6.3-STABLE after MFC of the `memrchr` function.
603104	June 15, 2008	6.3-STABLE after MFC of support for `:u` variable modifier in make(1).
604000	October 4, 2008	6.4-RELEASE
604100	October 4, 2008	6.4-STABLE after 6.4-RELEASE.
700000	July 11, 2005	7.0-CURRENT.
700001	July 23, 2005	7.0-CURRENT after bump of all shared library versions that had not been changed since RELENG_5.
700002	August 13, 2005	7.0-CURRENT after credential argument is added to dev_clone event handler.
700003	August 25, 2005	7.0-CURRENT after memmem(3) is added to libc.
700004	October 30, 2005	7.0-CURRENT after solisten(9) kernel arguments are modified to accept a backlog parameter.
700005	November 11, 2005	7.0-CURRENT after IFP2ENADDR() was changed to return a pointer to IF_LLADDR().
700006	November 11, 2005	7.0-CURRENT after addition of `if_addr` member to `struct ifnet` and IFP2ENADDR() removal.
700007	December 2, 2005	7.0-CURRENT after incorporating scripts from the local_startup directories into the base rcorder(8).
700008	December 5, 2005	7.0-CURRENT after removal of MNT_NODEV mount option.

Value	Date	Release
700009	December 19, 2005	7.0-CURRENT after ELF-64 type changes and symbol versioning.
700010	December 20, 2005	7.0-CURRENT after addition of hostb and vgapci drivers, addition of pci_find_extcap(), and changing the AGP drivers to no longer map the aperture.
700011	December 31, 2005	7.0-CURRENT after tv_sec was made time_t on all platforms but Alpha.
700012	January 8, 2006	7.0-CURRENT after ldconfig_local_dirs change.
700013	January 12, 2006	7.0-CURRENT after changes to `/etc/rc.d/abi` to support `/compat/linux/etc/ld.so.cache` being a symlink in a readonly filesystem.
700014	January 26, 2006	7.0-CURRENT after pts import.
700015	March 26, 2006	7.0-CURRENT after the introduction of version 2 of hwpmc(4)'s ABI.
700016	April 22, 2006	7.0-CURRENT after addition of fcloseall(3) to libc.
700017	May 13, 2006	7.0-CURRENT after removal of ip6fw.
700018	July 15, 2006	7.0-CURRENT after import of snd_emu10kx.
700019	July 29, 2006	7.0-CURRENT after import of OpenSSL 0.9.8b.
700020	September 3, 2006	7.0-CURRENT after addition of bus_dma_get_tag function
700021	September 4, 2006	7.0-CURRENT after libpcap 0.9.4 and tcpdump 3.9.4 import.

Value	Date	Release
700022	September 9, 2006	7.0-CURRENT after dlsym change to look for a requested symbol both in specified dso and its implicit dependencies.
700023	September 23, 2006	7.0-CURRENT after adding new sound IOCTLs for the OSSv4 mixer API.
700024	September 28, 2006	7.0-CURRENT after import of OpenSSL 0.9.8d.
700025	November 11, 2006	7.0-CURRENT after the addition of libelf.
700026	November 26, 2006	7.0-CURRENT after major changes on sound sysctls.
700027	November 30, 2006	7.0-CURRENT after the addition of Wi-Spy quirk.
700028	December 15, 2006	7.0-CURRENT after the addition of sctp calls to libc
700029	January 26, 2007	7.0-CURRENT after the GNU gzip(1) implementation was replaced with a BSD licensed version ported from NetBSD.
700030	February 7, 2007	7.0-CURRENT after the removal of IPIP tunnel encapsulation (VIFF_TUNNEL) from the IPv4 multicast forwarding code.
700031	February 23, 2007	7.0-CURRENT after the modification of bus_setup_intr() (newbus).
700032	March 2, 2007	7.0-CURRENT after the inclusion of ipw(4) and iwi(4) firmware.
700033	March 9, 2007	7.0-CURRENT after the inclusion of ncurses wide character support.
700034	March 19, 2007	7.0-CURRENT after changes to how insmntque(), get-

Value	Date	Release
		newvnode(), and vfs_hash_insert() work.
700035	March 26, 2007	7.0-CURRENT after addition of a notify mechanism for CPU frequency changes.
700036	April 6, 2007	7.0-CURRENT after import of the ZFS filesystem.
700037	April 8, 2007	7.0-CURRENT after addition of CAM 'SG' peripheral device, which implements a subset of Linux SCSI SG passthrough device API.
700038	April 30, 2007	7.0-CURRENT after changing getenv(3), putenv(3), setenv(3) and unsetenv(3) to be POSIX conformant.
700039	May 1, 2007	7.0-CURRENT after the changes in 700038 were backed out.
700040	May 10, 2007	7.0-CURRENT after the addition of flopen(3) to libutil.
700041	May 13, 2007	7.0-CURRENT after enabling symbol versioning, and changing the default thread library to libthr.
700042	May 19, 2007	7.0-CURRENT after the import of gcc 4.2.0.
700043	May 21, 2007	7.0-CURRENT after bump of all shared library versions that had not been changed since RELENG_6.
700044	June 7, 2007	7.0-CURRENT after changing the argument for vn_open()/VOP_OPEN() from file descriptor index to the struct file *.
700045	June 10, 2007	7.0-CURRENT after changing pam_nologin(8) to provide an account manage-

Value	Date	Release
		ment function instead of an authentication function to the PAM framework.
700046	June 11, 2007	7.0-CURRENT after updated 802.11 wireless support.
700047	June 11, 2007	7.0-CURRENT after adding TCP LRO interface capabilities.
700048	June 12, 2007	7.0-CURRENT after RFC 3678 API support added to the IPv4 stack. Legacy RFC 1724 behavior of the IP_MULTICAST_IF ioctl has now been removed; 0.0.0.0/8 may no longer be used to specify an interface index. struct ipmreqn should be used instead.
700049	July 3, 2007	7.0-CURRENT after importing pf from OpenBSD 4.1
(not changed)		7.0-CURRENT after adding IPv6 support for FAST_IPSEC, deleting KAME IPSEC, and renaming FAST_IPSEC to IPSEC.
700050	July 4, 2007	7.0-CURRENT after converting setenv/putenv/etc. calls from traditional BSD to POSIX.
700051	July 4, 2007	7.0-CURRENT after adding new mmap/lseek/etc syscalls.
700052	July 6, 2007	7.0-CURRENT after moving I4B headers to include/i4b.
700053	September 30, 2007	7.0-CURRENT after the addition of support for PCI domains

Value	Date	Release
700054	October 25, 2007	7.0-CURRENT after MFC of wide and single byte ctype separation.
700055	October 28, 2007	7.0-RELEASE, and 7.0-CURRENT after ABI backwards compatibility to the FreeBSD 4/5/6 versions of the PCIOCGETCONF, PCIOCREAD and PCIOCWRITE IOCTLs was MFCed, which required the ABI of the PCIOCGETCONF IOCTL to be broken again
700100	December 22, 2007	7.0-STABLE after 7.0-RELEASE
700101	February 8, 2008	7.0-STABLE after the MFC of m_collapse().
700102	March 30, 2008	7.0-STABLE after the MFC of kdb_enter_why().
700103	April 10, 2008	7.0-STABLE after adding l_sysid to struct flock.
700104	April 11, 2008	7.0-STABLE after the MFC of procstat(1).
700105	April 11, 2008	7.0-STABLE after the MFC of umtx features.
700106	April 15, 2008	7.0-STABLE after the MFC of write(2) support to psm(4).
700107	April 20, 2008	7.0-STABLE after the MFC of F_DUP2FD command to fcntl(2).
700108	May 5, 2008	7.0-STABLE after some lockmgr(9) changes, which makes it necessary to include `sys/lock.h` in order to use lockmgr(9).
700109	May 27, 2008	7.0-STABLE after MFC of the `memrchr` function.

Value	Date	Release
700110	August 5, 2008	7.0-STABLE after MFC of kernel NFS lockd client.
700111	August 20, 2008	7.0-STABLE after addition of physically contiguous jumbo frame support.
700112	August 27, 2008	7.0-STABLE after MFC of kernel DTrace support.
701000	November 25, 2008	7.1-RELEASE
701100	November 25, 2008	7.1-STABLE after 7.1-RELEASE.
701101	January 10, 2009	7.1-STABLE after `strndup` merge.
701102	January 17, 2009	7.1-STABLE after cpuctl(4) support added.
701103	February 7, 2009	7.1-STABLE after the merge of multi-/no-IPv4/v6 jails.
701104	February 14, 2009	7.1-STABLE after the store of the suspension owner in the struct mount, and introduction of vfs_susp_clean method into the struct vfsops.
701105	March 12, 2009	7.1-STABLE after the incompatible change to the kern.ipc.shmsegs sysctl to allow to allocate larger SysV shared memory segments on 64bit architectures.
701106	March 14, 2009	7.1-STABLE after the merge of a fix for POSIX semaphore wait operations.
702000	April 15, 2009	7.2-RELEASE
702100	April 15, 2009	7.2-STABLE after 7.2-RELEASE.
702101	May 15, 2009	7.2-STABLE after ichsmb(4) was changed to use left-adjusted slave addressing to

Value	Date	Release
		match other SMBus controller drivers.
702102	May 28, 2009	7.2-STABLE after MFC of the `fdopendir` function.
702103	June 06, 2009	7.2-STABLE after MFC of PmcTools.
702104	July 14, 2009	7.2-STABLE after MFC of the `closefrom` system call.
702105	July 31, 2009	7.2-STABLE after MFC of the SYSVIPC ABI change.
702106	September 14, 2009	7.2-STABLE after MFC of the x86 PAT enhancements and addition of d_mmap_single() and the scatter/gather list VM object type.
703000	February 9, 2010	7.3-RELEASE
703100	February 9, 2010	7.3-STABLE after 7.3-RELEASE.
704000	December 22, 2010	7.4-RELEASE
704100	December 22, 2010	7.4-STABLE after 7.4-RELEASE.
800000	October 11, 2007	8.0-CURRENT. Separating wide and single byte ctype.
800001	October 16, 2007	8.0-CURRENT after libpcap 0.9.8 and tcpdump 3.9.8 import.
800002	October 21, 2007	8.0-CURRENT after renaming kthread_create() and friends to kproc_create() etc.
800003	October 24, 2007	8.0-CURRENT after ABI backwards compatibility to the FreeBSD 4/5/6 versions of the PCIOCGETCONF, PCIOCREAD and PCIOCWRITE IOCTLs was added, which required the ABI of the PCIOCGETCONF IOCTL to be broken again

Value	Date	Release
800004	November 12, 2007	8.0-CURRENT after agp(4) driver moved from src/sys/pci to src/sys/dev/agp
800005	December 4, 2007	8.0-CURRENT after changes to the jumbo frame allocator (rev <revnumber>174247</revnumber>).
800006	December 7, 2007	8.0-CURRENT after the addition of callgraph capture functionality to hwpmc(4).
800007	December 25, 2007	8.0-CURRENT after kdb_enter() gains a "why" argument.
800008	December 28, 2007	8.0-CURRENT after LK_EXCLUPGRADE option removal.
800009	January 9, 2008	8.0-CURRENT after introduction of lockmgr_disown(9)
800010	January 10, 2008	8.0-CURRENT after the vn_lock(9) prototype change.
800011	January 13, 2008	8.0-CURRENT after the VOP_LOCK(9) and VOP_UNLOCK(9) prototype changes.
800012	January 19, 2008	8.0-CURRENT after introduction of lockmgr_recursed(9), BUF_RECURSED(9) and BUF_ISLOCKED(9) and the removal of `BUF_REFCNT()`.
800013	January 23, 2008	8.0-CURRENT after introduction of the “ASCII” encoding.
800014	January 24, 2008	8.0-CURRENT after changing the prototype of lockmgr(9)

Value	Date	Release
		and removal of `lockcount()` and `LOCKMGR_ASSERT()` .
800015	January 26, 2008	8.0-CURRENT after extending the types of the fts(3) structures.
800016	February 1, 2008	8.0-CURRENT after adding an argument to MEXTADD(9)
800017	February 6, 2008	8.0-CURRENT after the introduction of LK_NODUP and LK_NOWITNESS options in the lockmgr(9) space.
800018	February 8, 2008	8.0-CURRENT after the addition of m_collapse.
800019	February 9, 2008	8.0-CURRENT after the addition of current working directory, root directory, and jail directory support to the kern.proc.filedesc sysctl.
800020	February 13, 2008	8.0-CURRENT after introduction of lockmgr_assert(9) and `BUF_ASSERT` functions.
800021	February 15, 2008	8.0-CURRENT after introduction of lockmgr_args(9) and LK_INTERNAL flag removal.
800022	(backed out)	8.0-CURRENT after changing the default system ar to BSD ar(1).
800023	February 25, 2008	8.0-CURRENT after changing the prototypes of lockstatus(9) and VOP_ISLOCKED(9), more specifically retiring the `struct thread` argument.
800024	March 1, 2008	8.0-CURRENT after axing out the `lockwaiters` and `BUF_LOCKWAITERS` functions, changing the re-

Value	Date	Release
		turn value of brelvp from void to int and introducing new flags for lockinit(9).
800025	March 8, 2008	8.0-CURRENT after adding F_DUP2FD command to fcntl(2).
800026	March 12, 2008	8.0-CURRENT after changing the priority parameter to cv_broadcastpri such that 0 means no priority.
800027	March 24, 2008	8.0-CURRENT after changing the bpf monitoring ABI when zerocopy bpf buffers were added.
800028	March 26, 2008	8.0-CURRENT after adding l_sysid to struct flock.
800029	March 28, 2008	8.0-CURRENT after reintegration of the BUF_LOCKWAITERS function and the addition of lockmgr_waiters(9).
800030	April 1, 2008	8.0-CURRENT after the introduction of the rw_try_rlock(9) and rw_try_wlock(9) functions.
800031	April 6, 2008	8.0-CURRENT after the introduction of the lockmgr_rw and lockmgr_args_rw functions.
800032	April 8, 2008	8.0-CURRENT after the implementation of the openat and related syscalls, introduction of the O_EXEC flag for the open(2), and providing the corresponding linux compatibility syscalls.
800033	April 8, 2008	8.0-CURRENT after added write(2) support for psm(4)

Value	Date	Release
		in native operation level. Now arbitrary commands can be written to `/dev/psm%d` and status can be read back from it.
800034	April 10, 2008	8.0-CURRENT after introduction of the `memrchr` function.
800035	April 16, 2008	8.0-CURRENT after introduction of the `fdopendir` function.
800036	April 20, 2008	8.0-CURRENT after switchover of 802.11 wireless to multi-bss support (aka vaps).
800037	May 9, 2008	8.0-CURRENT after addition of multi routing table support (aka setfib(1), setfib(2)).
800038	May 26, 2008	8.0-CURRENT after removal of netatm and ISDN4BSD. Also, the addition of the Compact C Type (CTF) tools.
800039	June 14, 2008	8.0-CURRENT after removal of sgtty.
800040	June 26, 2008	8.0-CURRENT with kernel NFS lockd client.
800041	July 22, 2008	8.0-CURRENT after addition of arc4random_buf(3) and arc4random_uniform(3).
800042	August 8, 2008	8.0-CURRENT after addition of cpuctl(4).
800043	August 13, 2008	8.0-CURRENT after changing bpf(4) to use a single device node, instead of device cloning.
800044	August 17, 2008	8.0-CURRENT after the commit of the first step of the vimage project renaming global variables to be

Value	Date	Release
		virtualized with a V_ prefix with macros to map them back to their global names.
800045	August 20, 2008	8.0-CURRENT after the integration of the MPSAFE TTY layer, including changes to various drivers and utilities that interact with it.
800046	September 8, 2008	8.0-CURRENT after the separation of the GDT per CPU on amd64 architecture.
800047	September 10, 2008	8.0-CURRENT after removal of VSVTX, VSGID and VSUID.
800048	September 16, 2008	8.0-CURRENT after converting the kernel NFS mount code to accept individual mount options in the nmount() iovec, not just one big struct nfs_args.
800049	September 17, 2008	8.0-CURRENT after the removal of suser(9) and suser_cred(9).
800050	October 20, 2008	8.0-CURRENT after buffer cache API change.
800051	October 23, 2008	8.0-CURRENT after the removal of the MALLOC(9) and FREE(9) macros.
800052	October 28, 2008	8.0-CURRENT after the introduction of accmode_t and renaming of VOP_ACCESS 'a_mode' argument to 'a_accmode'.
800053	November 2, 2008	8.0-CURRENT after the prototype change of vfs_busy(9) and the introduction of its MBF_NOWAIT and MBF_MNTLSTLOCK flags.

Value	Date	Release
800054	November 22, 2008	8.0-CURRENT after the addition of buf_ring, memory barriers and ifnet functions to facilitate multiple hardware transmit queues for cards that support them, and a lockless ring-buffer implementation to enable drivers to more efficiently manage queuing of packets.
800055	November 27, 2008	8.0-CURRENT after the addition of Intel™ Core, Core2, and Atom support to hwpmc(4).
800056	November 29, 2008	8.0-CURRENT after the introduction of multi-/no-IPv4/v6 jails.
800057	December 1, 2008	8.0-CURRENT after the switch to the ath hal source code.
800058	December 12, 2008	8.0-CURRENT after the introduction of the VOP_VPTOCNP operation.
800059	December 15, 2008	8.0-CURRENT incorporates the new arp-v2 rewrite.
800060	December 19, 2008	8.0-CURRENT after the addition of makefs.
800061	January 15, 2009	8.0-CURRENT after TCP Appropriate Byte Counting.
800062	January 28, 2009	8.0-CURRENT after removal of minor(), minor2unit(), unit2minor(), etc.
800063	February 18, 2009	8.0-CURRENT after GENERIC config change to use the USB2 stack, but also the addition of fdevname(3).
800064	February 23, 2009	8.0-CURRENT after the USB2 stack is moved to and replaces dev/usb.

Value	Date	Release
800065	February 26, 2009	8.0-CURRENT after the renaming of all functions in libmp(3).
800066	February 27, 2009	8.0-CURRENT after changing USB devfs handling and layout.
800067	February 28, 2009	8.0-CURRENT after adding getdelim(), getline(), stpncpy(), strnlen(), wcsnlen(), wcscasecmp(), and wcsncasecmp().
800068	March 2, 2009	8.0-CURRENT after renaming the ushub devclass to uhub.
800069	March 9, 2009	8.0-CURRENT after libusb20.so.1 was renamed to libusb.so.1.
800070	March 9, 2009	8.0-CURRENT after merging IGMPv3 and Source-Specific Multicast (SSM) to the IPv4 stack.
800071	March 14, 2009	8.0-CURRENT after gcc was patched to use C99 inline semantics in c99 and gnu99 mode.
800072	March 15, 2009	8.0-CURRENT after the IFF_NEEDSGIANT flag has been removed; non-MPSAFE network device drivers are no longer supported.
800073	March 18, 2009	8.0-CURRENT after the dynamic string token substitution has been implemented for rpath and needed paths.
800074	March 24, 2009	8.0-CURRENT after tcpdump 4.0.0 and libpcap 1.0.0 import.
800075	April 6, 2009	8.0-CURRENT after layout of structs vnet_net, vnet_inet

Value	Date	Release
		and vnet_ipfw has been changed.
800076	April 9, 2009	8.0-CURRENT after adding delay profiles in dummynet.
800077	April 14, 2009	8.0-CURRENT after removing VOP_LEASE() and vop_vector.vop_lease.
800078	April 15, 2009	8.0-CURRENT after struct rt_weight fields have been added to struct rt_metrics and struct rt_metrics_lite, changing the layout of struct rt_metrics_lite. A bump to RTM_VERSION was made, but backed out.
800079	April 15, 2009	8.0-CURRENT after struct llentry pointers are added to struct route and struct route_in6.
800080	April 15, 2009	8.0-CURRENT after layout of struct inpcb has been changed.
800081	April 19, 2009	8.0-CURRENT after the layout of struct malloc_type has been changed.
800082	April 21, 2009	8.0-CURRENT after the layout of struct ifnet has changed, and with if_ref() and if_rele() ifnet refcounting.
800083	April 22, 2009	8.0-CURRENT after the implementation of a low-level Bluetooth HCI API.
800084	April 29, 2009	8.0-CURRENT after IPv6 SSM and MLDv2 changes.
800085	April 30, 2009	8.0-CURRENT after enabling support for VIMAGE kernel builds with one active image.

Value	Date	Release
800086	May 8, 2009	8.0-CURRENT after adding support for input lines of arbitrarily length in patch(1).
800087	May 11, 2009	8.0-CURRENT after some VFS KPI changes. The thread argument has been removed from the FSD parts of the VFS. `VFS_*` functions do not need the context any more because it always refers to `curthread`. In some special cases, the old behavior is retained.
800088	May 20, 2009	8.0-CURRENT after net80211 monitor mode changes.
800089	May 23, 2009	8.0-CURRENT after adding UDP control block support.
800090	May 23, 2009	8.0-CURRENT after virtualizing interface cloning.
800091	May 27, 2009	8.0-CURRENT after adding hierarchical jails and removing global securelevel.
800092	May 29, 2009	8.0-CURRENT after changing `sx_init_flags()` KPI. The `SX_ADAPTIVESPIN` is retired and a new `SX_NOADAPTIVE` flag is introduced in order to handle the reversed logic.
800093	May 29, 2009	8.0-CURRENT after adding mnt_xflag to struct mount.
800094	May 30, 2009	8.0-CURRENT after adding VOP_ACCESSX(9).
800095	May 30, 2009	8.0-CURRENT after changing the polling KPI. The polling handlers now return the number of packets processed. A new `IFCAP_POLLING_NOCOUNT` is also introduced to specify

Value	Date	Release
		that the return value is not significant and the counting should be skipped.
800096	June 1, 2009	8.0-CURRENT after updating to the new netisr implementation and after changing the way we store and access FIBs.
800097	June 8, 2009	8.0-CURRENT after the introduction of vnet destructor hooks and infrastructure.
800097	June 11, 2009	8.0-CURRENT after the introduction of netgraph outbound to inbound path call detection and queuing, which also changed the layout of struct thread.
800098	June 14, 2009	8.0-CURRENT after OpenSSL 0.9.8k import.
800099	June 22, 2009	8.0-CURRENT after NGROUPS update and moving route virtualization into its own VImage module.
800100	June 24, 2009	8.0-CURRENT after SYSVIPC ABI change.
800101	June 29, 2009	8.0-CURRENT after the removal of the /dev/net/* per-interface character devices.
800102	July 12, 2009	8.0-CURRENT after padding was added to struct sackhint, struct tcpcb, and struct tcpstat.
800103	July 13, 2009	8.0-CURRENT after replacing struct tcpopt with struct toeopt in the TOE driver interface to the TCP syncache.

Value	Date	Release
800104	July 14, 2009	8.0-CURRENT after the addition of the linker-set based per-vnet allocator.
800105	July 19, 2009	8.0-CURRENT after version bump for all shared libraries that do not have symbol versioning turned on.
800106	July 24, 2009	8.0-CURRENT after introduction of OBJT_SG VM object type.
800107	August 2, 2009	8.0-CURRENT after making the newbus subsystem Giant free by adding the newbus sxlock and 8.0-RELEASE.
800108	November 21, 2009	8.0-STABLE after implementing EVFILT_USER kevent filter.
800500	January 7, 2010	8.0-STABLE after `__FreeBSD_version` bump to make `pkg_add -r` use packages-8-stable.
800501	January 24, 2010	8.0-STABLE after change of the `scandir(3)` and `alphasort(3)` prototypes to conform to SUSv4.
800502	January 31, 2010	8.0-STABLE after addition of `sigpause(3)`.
800503	February 25, 2010	8.0-STABLE after addition of SIOCGIFDESCR and SIOCSIFDESCR ioctls to network interfaces. These ioctl can be used to manipulate interface description, as inspired by OpenBSD.
800504	March 1, 2010	8.0-STABLE after MFC of importing x86emu, a software emulator for real mode x86 CPU from OpenBSD.

Value	Date	Release
800505	May 18, 2010	8.0-STABLE after MFC of adding liblzma, xz, xzdec, and lzmainfo.
801000	June 14, 2010	8.1-RELEASE
801500	June 14, 2010	8.1-STABLE after 8.1-RELEASE.
801501	November 3, 2010	8.1-STABLE after KBI change in struct sysentvec, and implementation of PL_FLAG_SCE/SCX/EXEC/SI and pl_siginfo for ptrace(PT_LWPINFO) .
802000	December 22, 2010	8.2-RELEASE
802500	December 22, 2010	8.2-STABLE after 8.2-RELEASE.
802501	February 28, 2011	8.2-STABLE after merging DTrace changes, including support for userland tracing.
802502	March 6, 2011	8.2-STABLE after merging log2 and log2f into libm.
802503	May 1, 2011	8.2-STABLE after upgrade of the gcc to the last GPLv2 version from the FSF gcc-4_2-branch.
802504	May 28, 2011	8.2-STABLE after introduction of the KPI and supporting infrastructure for modular congestion control.
802505	May 28, 2011	8.2-STABLE after introduction of Hhook and Khelp KPIs.
802506	May 28, 2011	8.2-STABLE after addition of OSD to struct tcpcb.
802507	June 6, 2011	8.2-STABLE after ZFS v28 import.
802508	June 8, 2011	8.2-STABLE after removal of the schedtail event

Value	Date	Release
		handler and addition of the sv_schedtail method to struct sysvec.
802509	July 14, 2011	8.2-STABLE after merging the SSSE3 support into binutils.
802510	July 19, 2011	8.2-STABLE after addition of RFTSIGZMB flag for `rfork(2)`.
802511	September 9, 2011	8.2-STABLE after addition of automatic detection of USB mass storage devices which do not support the no synchronize cache SCSI command.
802512	September 10, 2011	8.2-STABLE after merging of re-factoring of auto-quirk.
802513	October 25, 2011	8.2-STABLE after merging of the MAP_PREFAULT_READ flag to `mmap(2)`.
802514	November 16, 2011	8.2-STABLE after merging of addition of posix_fallocate(2) syscall.
802515	January 6, 2012	8.2-STABLE after merging of addition of the posix_fadvise(2) system call.
802516	January 16, 2012	8.2-STABLE after merging gperf 3.0.3
802517	February 15, 2012	8.2-STABLE after introduction of the new extensible sysctl(3) interface NET_RT_IFLISTL to query address lists (rev <revnumber>231769</revnumber>).
803000	March 3, 2012	8.3-RELEASE.

Value	Date	Release
803500	March 3, 2012	8.3-STABLE after branching releng/8.3 (RELENG_8_3).
804000	March 28, 2013	8.4-RELEASE.
804500	March 28, 2013	8.4-STABLE after 8.4-RELEASE.
900000	August 22, 2009	9.0-CURRENT.
900001	September 8, 2009	9.0-CURRENT after importing x86emu, a software emulator for real mode x86 CPU from OpenBSD.
900002	September 23, 2009	9.0-CURRENT after implementing the EVFILT_USER kevent filter functionality.
900003	December 2, 2009	9.0-CURRENT after addition of `sigpause(3)` and PIE support in csu.
900004	December 6, 2009	9.0-CURRENT after addition of libulog and its libutempter compatibility interface.
900005	December 12, 2009	9.0-CURRENT after addition of `sleepq_sleepcnt()`, which can be used to query the number of waiters on a specific waiting queue.
900006	January 4, 2010	9.0-CURRENT after change of the `scandir(3)` and `alphasort(3)` prototypes to conform to SUSv4.
900007	January 13, 2010	9.0-CURRENT after the removal of utmp(5) and the addition of utmpx (see `getutxent(3)`) for improved logging of user logins and system events.
900008	January 20, 2010	9.0-CURRENT after the import of BSDL bc/dc and the deprecation of GNU bc/dc.

Value	Date	Release
900009	January 26, 2010	9.0-CURRENT after the addition of SIOCGIFDESCR and SIOCSIFDESCR ioctls to network interfaces. These ioctl can be used to manipulate interface description, as inspired by OpenBSD.
900010	March 22, 2010	9.0-CURRENT after the import of zlib 1.2.4.
900011	April 24, 2010	9.0-CURRENT after adding soft-updates journalling.
900012	May 10, 2010	9.0-CURRENT after adding liblzma, xz, xzdec, and lzmainfo.
900013	May 24, 2010	9.0-CURRENT after bringing in USB fixes for linux(4).
900014	June 10, 2010	9.0-CURRENT after adding Clang.
900015	July 22, 2010	9.0-CURRENT after the import of BSD grep.
900016	July 28, 2010	9.0-CURRENT after adding mti_zone to struct malloc_type_internal.
900017	August 23, 2010	9.0-CURRENT after changing back default grep to GNU grep and adding WITH_BSD_GREP knob.
900018	August 24, 2010	9.0-CURRENT after the `pthread_kill(3)` -generated signal is identified as SI_LWP in si_code. Previously, si_code was SI_USER.
900019	August 28, 2010	9.0-CURRENT after addition of the MAP_PREFAULT_READ flag to `mmap(2)`.
900020	September 9, 2010	9.0-CURRENT after adding drain functionality to sbufs,

Value	Date	Release
		which also changed the layout of struct sbuf.
900021	September 13, 2010	9.0-CURRENT after DTrace has grown support for userland tracing.
900022	October 2, 2010	9.0-CURRENT after addition of the BSDL man utilities and retirement of GNU/GPL man utilities.
900023	October 11, 2010	9.0-CURRENT after updating xz to git 20101010 snapshot.
900024	November 11, 2010	9.0-CURRENT after libgcc.a was replaced by libcompiler_rt.a.
900025	November 12, 2010	9.0-CURRENT after the introduction of the modularised congestion control.
900026	November 30, 2010	9.0-CURRENT after the introduction of Serial Management Protocol (SMP) passthrough and the XPT_SMP_IO and XPT_GDEV_ADVINFO CAM CCBs.
900027	December 5, 2010	9.0-CURRENT after the addition of log2 to libm.
900028	December 21, 2010	9.0-CURRENT after the addition of the Hhook (Helper Hook), Khelp (Kernel Helpers) and Object Specific Data (OSD) KPIs.
900029	December 28, 2010	9.0-CURRENT after the modification of the TCP stack to allow Khelp modules to interact with it via helper hook points and store per-connection data in the TCP control block.

Value	Date	Release
900030	January 12, 2011	9.0-CURRENT after the update of libdialog to version 20100428.
900031	February 7, 2011	9.0-CURRENT after the addition of `pthread_getthreadid_np(3)` .
900032	February 8, 2011	9.0-CURRENT after the removal of the uio_yield prototype and symbol.
900033	February 18, 2011	9.0-CURRENT after the update of binutils to version 2.17.50.
900034	March 8, 2011	9.0-CURRENT after the struct sysvec (sv_schedtail) changes.
900035	March 29, 2011	9.0-CURRENT after the update of base gcc and libstdc++ to the last GPLv2 licensed revision.
900036	April 18, 2011	9.0-CURRENT after the removal of libobjc and Objective-C support from the base system.
900037	May 13, 2011	9.0-CURRENT after importing the libprocstat(3) library and fuser(1) utility to the base system.
900038	May 22, 2011	9.0-CURRENT after adding a lock flag argument to VFS_FHTOVP(9).
900039	June 28, 2011	9.0-CURRENT after importing pf from OpenBSD 4.5.
900040	July 19, 2011	Increase default MAXCPU for FreeBSD to 64 on amd64 and ia64 and to 128 for XLP (mips).
900041	August 13, 2011	9.0-CURRENT after the implementation of Capsicum

Value	Date	Release
		capabilities; fget(9) gains a rights argument.
900042	August 28, 2011	Bump shared libraries' version numbers for libraries whose ABI has changed in preparation for 9.0.
900043	September 2, 2011	Add automatic detection of USB mass storage devices which do not support the no synchronize cache SCSI command.
900044	September 10, 2011	Re-factor auto-quirk. 9.0-RELEASE.
900045	January 2, 2012	9-CURRENT after MFC of true/false from 1000002.
900500	January 2, 2012	9.0-STABLE.
900501	January 6, 2012	9.0-STABLE after merging of addition of the posix_fadvise(2) system call.
900502	January 16, 2012	9.0-STABLE after merging gperf 3.0.3
900503	February 15, 2012	9.0-STABLE after introduction of the new extensible sysctl(3) interface NET_RT_IFLISTL to query address lists (rev <revnumber>231768</revnumber>).
900504	March 3, 2012	9.0-STABLE after changes related to mounting of filesystem inside a jail (rev <revnumber>232728</revnumber>).
900505	March 13, 2012	9.0-STABLE after introduction of new tcp(4) socket options: TCP_KEEPINIT, TCP_KEEPIDLE,

Value	Date	Release
		TCP_KEEPINTVL, and TCP_KEEPCNT (rev <revnumber>232945</revnumber>).
900506	May 22, 2012	9.0-STABLE after introduction of the quick_exit function and related changes required for C++11 (rev <revnumber>235786</revnumber>).
901000	August 5, 2012	9.1-RELEASE.
901500	August 6, 2012	9.1-STABLE after branching releng/9.1 (RELENG_9_1).
901501	November 11, 2012	9.1-STABLE after LIST_PREV() added to queue.h (rev <revnumber>242893</revnumber>) and KBI change in USB serial devices (rev <revnumber>240659</revnumber>).
901502	November 28, 2012	9.1-STABLE after USB serial jitter buffer requires rebuild of USB serial device modules.
901503	February 21, 2013	9.1-STABLE after USB moved to the driver structure requiring a rebuild of all USB modules. Also indicates the presence of nmtree.
901504	March 15, 2013	9.1-STABLE after install gained -l, -M, -N and related flags and cat gained the -l option.

Value	Date	Release
901505	June 13, 2013	9.1-STABLE after fixes in ctfmerge bootstrapping (rev <revnumber>249243</revnumber>).
902001	August 3, 2013	`releng/9.2` branched from `stable/9` (rev <revnumber>253912</revnumber>).
902501	August 2, 2013	9.2-STABLE after creation of `releng/9.2` branch (rev <revnumber>253913</revnumber>).
1000000	September 26, 2011	10.0-CURRENT.
1000001	November 4, 2011	10-CURRENT after addition of the posix_fadvise(2) system call.
1000002	December 12, 2011	10-CURRENT after defining boolean true/false in sys/types.h, sizeof(bool) may have changed (rev <revnumber>228444</revnumber>). 10-CURRENT after xlocale.h was introduced (rev <revnumber>227753</revnumber>).
1000003	December 16, 2011	10-CURRENT after major changes to carp(4), changing size of struct in_aliasreq, struct in6_aliasreq (rev <revnumber>228571</revnumber>) and straitening arguments check of SIOCAIFADDR (rev

Value	Date	Release
		<revnumber>228574</revnumber>).
1000004	January 1, 2012	10-CURRENT after the removal of skpc(9) and the addition of memcchr(9) (rev <revnumber>229200</revnumber>).
1000005	January 16, 2012	10-CURRENT after the removal of support for SIOCSIFADDR, SIOCSIFNETMASK, SIOCSIFBRDADDR, SIOCSIFDSTADDR ioctls (rev <revnumber>230207</revnumber>).
1000006	January 26, 2012	10-CURRENT after introduction of read capacity data asynchronous notification in the cam(4) layer (rev <revnumber>230590</revnumber>).
1000007	February 5, 2012	10-CURRENT after introduction of new tcp(4) socket options: TCP_KEEPINIT, TCP_KEEPIDLE, TCP_KEEPINTVL, and TCP_KEEPCNT (rev <revnumber>231025</revnumber>).
1000008	February 11, 2012	10-CURRENT after introduction of the new extensible sysctl(3) interface NET_RT_IFLISTL to query address lists (rev <revnumber>231505</revnumber>).

Value	Date	Release
1000009	February 25, 2012	10-CURRENT after import of libarchive 3.0.3 (rev <revnumber>232153</revnumber>).
1000010	March 31, 2012	10-CURRENT after xlocale cleanup (rev <revnumber>233757</revnumber>).
1000011	April 16, 2012	10-CURRENT import of LLVM/Clang 3.1 trunk r154661 (rev <revnumber>234353</revnumber>).
1000012	May 2, 2012	10-CURRENT jemalloc import (rev <revnumber>234924</revnumber>).
1000013	May 22, 2012	10-CURRENT after byacc import (rev <revnumber>235788</revnumber>).
1000014	June 27, 2012	10-CURRENT after BSD sort becoming the default sort (rev <revnumber>237629</revnumber>).
1000015	July 12, 2012	10-CURRENT after import of OpenSSL 1.0.1c (rev <revnumber>238405</revnumber>).
(not changed)	July 13, 2012	10-CURRENT after the fix for LLVM/Clang 3.1 regression (rev

Value	Date	Release
		<revnumber>238429</revnumber>).
1000016	August 8, 2012	10-CURRENT after KBI change in ucom(4) (rev <revnumber>239179</revnumber>).
1000017	August 8, 2012	10-CURRENT after adding streams feature to the USB stack (rev <revnumber>239214</revnumber>).
1000018	September 8, 2012	10-CURRENT after major rewrite of pf(4) (rev <revnumber>240233</revnumber>).
1000019	October 6, 2012	10-CURRENT after pfil(9) KBI/KPI changed to supply packets in net byte order to AF_INET filter hooks (rev <revnumber>241245</revnumber>).
1000020	October 16, 2012	10-CURRENT after the network interface cloning KPI changed and struct if_clone becoming opaque (rev <revnumber>241610</revnumber>).
1000021	October 22, 2012	10-CURRENT after removal of support for non-MPSAFE filesystems and addition of support for FUSEFS (rev <revnumber>241519</revnumber> ,

Value	Date	Release
		<revnumber>241897</revnumber>).
1000022	October 22, 2012	10-CURRENT after the entire IPv4 stack switched to network byte order for IP packet header storage (rev <revnumber>241913</revnumber>).
1000023	November 5, 2012	10-CURRENT after jitter buffer in the common USB serial driver code, to temporarily store characters if the TTY buffer is full. Add flow stop and start signals when this happens (rev <revnumber>242619</revnumber>).
1000024	November 5, 2012	10-CURRENT after clang was made the default compiler on i386 and amd64 (rev <revnumber>242624</revnumber>).
1000025	November 17, 2012	10-CURRENT after the sin6_scope_id member variable in struct sockaddr_in6 was changed to being filled by the kernel before passing the structure to the userland via sysctl or routing socket. This means the KAME-specific embedded scope id in sin6_addr.s6_addr[2] is always cleared in userland application (rev <revnumber>243443</revnumber>).

Value	Date	Release
1000026	January 11, 2013	10-CURRENT after install gained the -N flag (rev <revnumber>245313</revnumber>). May also be used to indicate the presence of nmtree.
1000027	January 29, 2013	10-CURRENT after cat gained the -l flag (rev <revnumber>246083</revnumber>).
1000028	February 13, 2013	10-CURRENT after USB moved to the driver structure requiring a rebuild of all USB modules (rev <revnumber>246759</revnumber>).
1000029	March 4, 2013	10-CURRENT after the introduction of tickless callout facility which also changed the layout of struct callout (rev <revnumber>247777</revnumber>).
1000030	March 12, 2013	10-CURRENT after KPI breakage introduced in the VM subsystem to support read/write locking (rev <revnumber>248084</revnumber>).
1000031	April 26, 2013	10-CURRENT after the dst parameter of the ifnet `if_output` method was changed to take const qualifier (rev <revnumber>249925</revnumber>

Value	Date	Release
		).
1000032	May 1, 2013	10-CURRENT after the introduction of the `accept4` (rev <revnumber>250154</revnumber>) and `pipe2` (rev <revnumber>250159</revnumber>) system calls.
1000033	May 21, 2013	10-CURRENT after flex 2.5.37 import (rev <revnumber>250881</revnumber>).
1000034	June 3, 2013	10-CURRENT after the addition of the following functions to libm: `cacos`, `cacosf`, `cacosh`, `cacoshf`, `casin`, `casinf`, `casinh`, `casinhf`, `catan`, `catanf`, `catanh`, `catanhf`, `logl`, `log2l`, `log10l`, `log1pl`, `expm1l` (rev <revnumber>251294</revnumber>).
1000035	June 8, 2013	10-CURRENT after the introduction of the `aio_mlock` system call (rev <revnumber>251526</revnumber>).
1000036	July 9, 2013	10-CURRENT after the addition of a new function to the kernel GSSAPI module's function call interface (rev <revnumber>253049</revnumber>).

Value	Date	Release
1000037	July 9, 2013	10-CURRENT after the migration of statistics structures to PCPU counters. Changed structures include: ahstat, arpstat, espstat, icmp6_ifstat , icmp6stat , in6_ifstat , ip6stat , ipcompstat, ipipstat, ipsecstat, mrt6stat , mrtstat, pfkeystat, pim6stat , pimstat, rip6stat , udpstat (rev <revnumber>253081</revnumber>).
1000038	July 16, 2013	10-CURRENT after making ARM EABI the default ABI on arm, armeb, armv6, and armv6eb architectures (rev <revnumber>253396</revnumber>).
1000039	July 22, 2013	10-CURRENT after CAM and mps(4) driver scanning changes (rev <revnumber>253549</revnumber>).
1000040	July 24, 2013	10-CURRENT after addition of libusb pkgconf files (rev <revnumber>253638</revnumber>).
1000041	August 5, 2013	10-CURRENT after change from time_second to time_uptime in PF_INET6 (rev <revnumber>253970</revnumber>).

Value	Date	Release
1000042	August 9, 2013	10-CURRENT after VM subsystem change to unify soft and hard busy mechanisms (rev <revnumber>254138</revnumber>).
1000043	August 13, 2013	10-CURRENT after `WITH_ICONV` is enabled by default. A new src.conf(5) option, `WITH_LIBICONV_COMPAT` (disabled by default) adds `libiconv_open` to provide compatibility with the libiconv port (rev <revnumber>254273</revnumber>).
1000044	August 15, 2013	10-CURRENT after `libc.so` conversion to an ld(1) script (rev <revnumber>251668</revnumber> , <revnumber>254358</revnumber>).
1000045	August 15, 2013	10-CURRENT after devfs programming interface change by replacing the cdevsw flag `D_UNMAPPED_IO` with the struct cdev flag `SI_UNMAPPED` (rev <revnumber>254389</revnumber>).
1000046	August 19, 2013	10-CURRENT after addition of `M_PROTO[9-12]` and removal of `M_FRAG|`

Value	Date	Release
		M_FIRSTFRAG|M_LASTFRAG mbuf flags (rev <revnumber>254524</revnumber>, <revnumber>254526</revnumber>).
1000047	August 21, 2013	10-CURRENT after stat(2) update to allow storing some Windows/DOS and CIFS file attributes as stat(2) flags (rev <revnumber>254627</revnumber>).
1000048	August 22, 2013	10-CURRENT after modification of structure xsctp_inpcb (rev <revnumber>254672</revnumber>).
1000049	August 24, 2013	10-CURRENT after physio(9) support for devices that do not function properly with split I/O, such as sa(4) (rev <revnumber>254760</revnumber>).
1000050	August 24, 2013	10-CURRENT after modifications of structure mbuf (rev <revnumber>254780</revnumber>, <revnumber>254799</revnumber>, <revnumber>254804</revnumber>,

Value	Date	Release
		<revnumber>254807</revnumber> <revnumber>254842</revnumber>).
1000051	August 25, 2013	10-CURRENT after Radeon KMS driver import (rev <revnumber>254885</revnumber> , <revnumber>254887</revnumber>).
1000052	September 3, 2013	10-CURRENT after import of NetBSD `libexecinfo` is connected to the build (rev <revnumber>255180</revnumber>).
1000053	September 6, 2013	10-CURRENT after API and ABI changes to the Capsicum framework (rev <revnumber>255305</revnumber>).
1000054	September 6, 2013	10-CURRENT after `gcc` and `libstdc++` are no longer built by default (rev <revnumber>255321</revnumber>).
1000055	September 6, 2013	10-CURRENT after addition of `MMAP_32BIT` mmap(2) flag (rev <revnumber>255426</revnumber>).
1000500	October 10, 2013	10-STABLE after branch from `head/` (rev

Value	Date	Release
		<revnumber>256283</revnumber>).
1100000	October 10, 2013	11.0-CURRENT. (rev <revnumber>256284</revnumber>).

Note

Note that 2.2-STABLE sometimes identifies itself as "2.2.5-STABLE" after the 2.2.5-RELEASE. The pattern used to be year followed by the month, but we decided to change it to a more straightforward major/minor system starting from 2.2. This is because the parallel development on several branches made it infeasible to classify the releases simply by their real release dates. If you are making a port now, you do not have to worry about old -CURRENTs; they are listed here just for your reference.