

T@ix

FreeBSD Porter's Handbook

The FreeBSD Documentation Project

FreeBSD Porter's Handbook

by The FreeBSD Documentation Project

Published April 2000

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

FreeBSD is a registered trademark of The FreeBSD Foundation.

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

Sun, Sun Microsystems, SunOS, Solaris, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Apple and QuickTime are trademarks of Apple Computer, Inc., registered in the U.S. and other countries.

Macromedia and Flash are trademarks or registered trademarks of Macromedia, Inc. in the United States and/or other countries.

Microsoft, Windows, and Windows Media are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

PartitionMagic is a registered trademark of PowerQuest Corporation in the United States and/or 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 book, and the FreeBSD Project was aware of the trademark claim, the designations have been followed by the 'TM' symbol.

Redistribution and use in source (SGML DocBook) and 'compiled' forms (SGML, 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 (SGML 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.

Table of Contents

1 Introduction	??
2 Making a port yourself	??
3 Quick Porting	??
3.1 Writing the Makefile	??
3.2 Writing the description files	??
3.2.1 pkg-descr	??
3.2.2 pkg-plist	??
3.3 Creating the checksum file	??
3.4 Testing the port	??
3.5 Checking your port with portlint	??
3.6 Submitting the port	??
4 Slow Porting	??
4.1 How things work	??
4.2 Getting the original sources	??
4.3 Modifying the port	??
4.4 Patching	??
4.5 Configuring	??
4.6 Handling user input	??
5 Configuring the Makefile	??
5.1 The original source	??
5.2 Naming	??
5.2.1 PORTNAME and PORTVERSION	??
5.2.2 PORTREVISION and PORTEPOCH	??
5.2.3 PKGNAMEPREFIX and PKGNAMESUFFIX	??
5.2.4 LATEST_LINK	??
5.2.5 Package Naming Conventions	??
5.3 Categorization	??
5.3.1 CATEGORIES	??
5.3.2 Current list of categories	??
5.3.3 Choosing the right category	??
5.3.4 Proposing a new category	??
5.3.5 Proposing reorganizing all the categories	??
5.4 The distribution files	??
5.4.1 DISTVERSION/DISTNAME	??
5.4.2 MASTER_SITES	??
5.4.3 EXTRACT_SUFX	??
5.4.4 DISTFILES	??
5.4.5 EXTRACT_ONLY	??
5.4.6 PATCHFILES	??
5.4.7 Multiple distribution files or patches from different sites and subdirectories (MASTER_SITES:n)	??
5.4.8 DIST_SUBDIR	??
5.4.9 ALWAYS_KEEP_DISTFILES	??
5.5 MAINTAINER	??
5.6 COMMENT	??

5.7 Dependencies	??
5.7.1 LIB_DEPENDS.....	??
5.7.2 RUN_DEPENDS.....	??
5.7.3 BUILD_DEPENDS	??
5.7.4 FETCH_DEPENDS	??
5.7.5 EXTRACT_DEPENDS	??
5.7.6 PATCH_DEPENDS	??
5.7.7 USE_*.....	??
5.7.8 Minimal version of a dependency.....	??
5.7.9 Notes on dependencies	??
5.7.10 Circular dependencies are fatal.....	??
5.8 MASTERDIR	??
5.9 Manpages	??
5.10 Info files.....	??
5.11 Makefile Options	??
5.11.1 Knobs.....	??
5.11.2 OPTIONS	??
5.11.3 Feature auto-activation.....	??
5.12 Specifying the working directory	??
5.12.1 WRKSRC	??
5.12.2 NO_WRKSUBDIR	??
5.13 CONFLICTS	??
5.14 Installing files	??
5.14.1 INSTALL_* macros	??
5.14.2 Stripping Binaries	??
5.14.3 Installing a whole tree of files.....	??
5.14.4 Install additional documentation.....	??
5.14.5 Subdirectories under PREFIX	??
6 Special considerations.....	??
6.1 Shared Libraries	??
6.2 Ports with distribution restrictions	??
6.2.1 NO_PACKAGE.....	??
6.2.2 NO_CDROM.....	??
6.2.3 NOFETCHFILES	??
6.2.4 RESTRICTED.....	??
6.2.5 RESTRICTED_FILES.....	??
6.3 Building mechanisms	??
6.3.1 Parallel ports building.....	??
6.3.2 make, gmake, and imake	??
6.3.3 configure script.....	??
6.3.4 Using scons.....	??
6.4 Using GNU autotools	??
6.4.1 Introduction.....	??
6.4.2 libtool	??
6.4.3 libltdl	??
6.4.4 autoconf and autoheader.....	??
6.4.5 automake and aclocal	??

6.5 Using GNU <code>gettext</code>	??
6.5.1 Basic usage	??
6.5.2 Optional usage	??
6.5.3 Handling message catalog directories	??
6.6 Using <code>perl</code>	??
6.7 Using X11	??
6.7.1 X.Org components	??
6.7.2 Ports that require Motif	??
6.7.3 X11 fonts	??
6.7.4 Getting fake <code>DISPLAY</code> using <code>Xvfb</code>	??
6.7.5 Desktop entries	??
6.8 Using GNOME	??
6.9 Using KDE	??
6.9.1 Variable definitions	??
6.9.2 Ports that require Qt	??
6.9.3 Component selection (Qt 4.x only)	??
6.9.4 Additional considerations	??
6.10 Using Java	??
6.10.1 Variable definitions	??
6.10.2 Building with Ant	??
6.10.3 Best practices	??
6.11 Web applications, Apache and PHP	??
6.11.1 Apache	??
6.11.2 Web applications	??
6.11.3 PHP	??
6.11.4 PEAR modules	??
6.12 Using Python	??
6.13 Using Tcl/Tk	??
6.14 Using Emacs	??
6.15 Using Ruby	??
6.16 Using SDL	??
6.17 Using wxWidgets	??
6.17.1 Introduction	??
6.17.2 Version selection	??
6.17.3 Component selection	??
6.17.4 Unicode	??
6.17.5 Detecting installed versions	??
6.17.6 Defined variables	??
6.17.7 Processing in <code>bsd.port.pre.mk</code>	??
6.17.8 Additional <code>configure</code> arguments	??
6.18 Using Lua	??
6.18.1 Introduction	??
6.18.2 Version selection	??
6.18.3 Component selection	??
6.18.4 Detecting installed versions	??
6.18.5 Defined variables	??
6.18.6 Processing in <code>bsd.port.pre.mk</code>	??
6.19 Using Xfce	??

6.20 Using databases	??
6.21 Starting and stopping services (rc scripts).....	??
6.21.1 Stopping services at deinstall.....	??
7 Advanced pkg-plist practices.....	??
7.1 Changing pkg-plist based on make variables	??
7.2 Empty directories	??
7.2.1 Cleaning up empty directories	??
7.2.2 Creating empty directories.....	??
7.3 Configuration files	??
7.4 Dynamic vs. static package list	??
7.5 Automated package list creation	??
8 The pkg- * files.....	??
8.1 pkg-message	??
8.2 pkg-install	??
8.3 pkg-deinstall	??
8.4 pkg-req.....	??
8.5 Changing the names of pkg- * files.....	??
8.6 Making use of SUB_FILES and SUB_LIST	??
9 Testing your port.....	??
9.1 Running make describe	??
9.2 Portlint.....	??
9.3 Port Tools	??
9.4 PREFIX and DESTDIR	??
9.5 Tinderbox	??
10 Upgrading	??
11 Ports security	??
11.1 Why security is so important.....	??
11.2 Fixing security vulnerabilities	??
11.3 Keeping the community informed.....	??
11.3.1 The VuXML database.....	??
11.3.2 A short introduction to VuXML	??
11.3.3 Testing your changes to the VuXML database	??
12 Dos and Don'ts	??
12.1 Introduction	??
12.2 WRKDIR	??
12.3 WRKDIRPREFIX	??
12.4 Differentiating operating systems and OS versions	??
12.5 __FreeBSD_version values	??
12.6 Writing something after bsd.port.mk.....	??
12.7 Use the exec statement in wrapper scripts	??
12.8 UIDs and GIDs.....	??
12.9 Do things rationally	??
12.10 Respect both CC and CXX.....	??
12.11 Respect CFLAGS	??
12.12 Threading libraries	??
12.13 Feedback.....	??

12.14	<code>README.html</code>	??
12.15	Marking a port not installable with <code>BROKEN</code> , <code>FORBIDDEN</code> , or <code>IGNORE</code>	??
12.15.1	Variables	??
12.15.2	Implementation Notes.....	??
12.16	Marking a port for removal with <code>DEPRECATED</code> or <code>EXPIRATION_DATE</code>	??
12.17	Avoid use of the <code>.error</code> construct	??
12.18	Usage of <code>sysctl</code>	??
12.19	Rerolling distfiles	??
12.20	Necessary workarounds.....	??
12.21	Miscellanea.....	??
13	A Sample Makefile.....	??
14	Keeping Up	??
14.1	FreshPorts.....	??
14.2	The Web Interface to the Source Repository.....	??
14.3	The FreeBSD Ports Mailing List.....	??
14.4	The FreeBSD Port Building Cluster on <code>pointyhat.FreeBSD.org</code>	??
14.5	The FreeBSD Port Distfile Survey	??
14.6	The FreeBSD Ports Monitoring System.....	??

List of Tables

5-1. The <code>USE_*</code> variables	??
5-2. Common <code>WITH_*</code> and <code>WITHOUT_*</code> variables	??
6-1. Variables for ports related to <code>gmake</code>	??
6-2. Variables for ports that use <code>configure</code>	??
6-3. Variables for ports that use <code>scons</code>	??
6-4. Variables for ports that use <code>perl</code>	??
6-5. Variables for ports that use <code>X</code>	??
6-6. Variables for depending on individual parts of <code>X11</code>	??
6-7. Variables for ports that use <code>KDE</code>	??
6-8. Variables for ports that use <code>Qt</code>	??
6-9. Additional variables for ports that use <code>Qt 4.x</code>	??
6-10. Available <code>Qt4</code> library components	??
6-11. Available <code>Qt4</code> tool components	??
6-12. Available <code>Qt4</code> plugin components	??
6-13. Variables that may be set by ports that use <code>Java</code>	??
6-14. Variables provided to ports that use <code>Java</code>	??
6-15. Constants defined for ports that use <code>Java</code>	??
6-16. Variables for ports that use <code>Apache</code>	??
6-17. Useful variables for porting <code>Apache</code> modules	??
6-18. Variables for ports that use <code>PHP</code>	??
6-19. Most useful variables for ports that use <code>Python</code>	??
6-20. The most useful variables for ports that use <code>Tcl/Tk</code>	??
6-21. Useful variables for ports that use <code>Ruby</code>	??
6-22. Selected read-only variables for ports that use <code>Ruby</code>	??
6-23. Variables to select wxWidgets versions	??
6-24. Available wxWidgets versions	??
6-25. wxWidgets version specifications	??
6-26. Variables to select preferred wxWidgets versions	??
6-27. Available wxWidgets components	??
6-28. Available wxWidgets dependency types	??
6-29. Default wxWidgets dependency types	??
6-30. Variables to select Unicode in wxWidgets versions	??
6-31. Variables defined for ports that use wxWidgets	??
6-32. Legal values for <code>WX_CONF_ARGS</code>	??
6-33. Variables to select Lua versions	??
6-34. Available Lua versions	??
6-35. Lua version specifications	??
6-36. Variables to select preferred Lua versions	??
6-37. Available Lua components	??
6-38. Available Lua dependency types	??
6-39. Default Lua dependency types	??
6-40. Variables defined for ports that use Lua	??
6-41. Variables for ports using databases	??
12-1. <code>__FreeBSD_version</code> values	??

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 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.

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 (<http://lists.FreeBSD.org/mailman/listinfo/freebsd-ports>).

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.

Chapter 3 Quick Porting

This section tells you how to do a quick 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.

3.1 Writing the Makefile

The minimal Makefile would look something like this:

```
# New ports collection makefile for:  oneko
# Date created:          5 December 1994
# Whom:                  asami
#
# $FreeBSD$

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

MAINTAINER=    asami@FreeBSD.org
COMMENT=       A cat chasing a mouse all over the screen

MAN1=         oneko.1
MANCOMPRESSED= yes
USE_IMAKE=     yes

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

See if you can figure it out. Do not worry about the contents of the `$FreeBSD$` line, it will be filled in automatically by CVS 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). If the ported software has an official WWW homepage, you should list it here. Prefix *one* of the websites with `WWW:` so that automated tools will work correctly.

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` or `/usr/X11R6`). If you are using the `MANn` variables (as you should be), do not list any `manpages` here. 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
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 \
                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.

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 your port
- `pkg-plist` contains everything that is installed by your port
- Your port can be installed multiple times using the `reinstall` target
- Your port cleans up after itself upon deinstall

Recommended test ordering

1. `make install`
2. `make package`
3. `make deinstall`
4. `pkg_add package-name`
5. `make deinstall`
6. `make reinstall`
7. `make package`

Make sure that there are not any warnings issued in any of the `package` and `deinstall` stages. After step 3, check to see if all the new directories are correctly deleted. Also, try using the software after step 4, to ensure that it works correctly when installed from a package.

The most thorough way to automate these steps is via installing the **ports tinderbox**. This maintains `jails` in which you can test all of the above steps without changing the state of your running system. Please see `ports/ports-mgmt/tinderbox` for more information.

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 port

First, 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, simply include the output of `shar `find port_dir`` in a bug report and send it with the `send-pr(1)` program (see Bug Reports and General Commentary (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/contributing/contrib-how.html#CONTRIB-GENERAL) 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 and the `shar` 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 and “Update port: <category>/<portname> <short description of the update>” for port updates. 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`.*

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 waiting to be committed to FreeBSD (<http://www.FreeBSD.org/cgi/query-pr-summary.cgi?category=ports>).

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 appear in the list of Additional FreeBSD Contributors (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/contributors/contrib-additional.html) and other files. Isn't that great?!? :-)

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 main FTP site at `ftp://ftp.FreeBSD.org/pub/FreeBSD/ports/distfiles/`, where we put sanctioned 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'd 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.
 3. If `USE_IMAKE` is set, `XMKMF` (default: `xmkmf -a`) is run.
5. The `build` target is run. This is responsible for descending into the port's private working directory (`WRKSRC`) and building it. If `USE_GMAKE` is set, `GNU make` will be used, otherwise the system `make` will be used.

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!

Now that you understand what goes on when the user types `make`, 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.Z`) 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 `WRKSR`C (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 `WRKSR`C/`foobar.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 path and file name in patch names.

Do not put RCS strings in patches. CVS 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, `Makefiles` 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 Open Source world that projects share large amount of code base, but obey different style and indenting rules. If you take working piece of functionality from one project to fix similar area in another, please be careful: the resulting line patch may be full of non-functional changes. It does not only increase the size of the CVS repository but makes it hard to find out what had exactly caused the problem and what did you change 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' ${WRKSR}C/README
@${REINPLACE_CMD} -e 's|-pthread|${PTHREAD_LIBS}|' ${WRKSR}C/configure
```

Quite often, there is a situation when the software being ported, especially if it is primarily developed on Windows®, uses the CR/LF convention for most of its source files. This may cause problems with further patching, compiler warnings, scripts execution (`/bin/sh^M` not found), etc. To quickly convert all files from CR/LF to just LF, add `USE_DOS2UNIX=yes` to the port `Makefile`. A list of files to convert can be specified:

```
USE_DOS2UNIX=    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:

```
USE_DOS2UNIX=    yes
DOS2UNIX_REGEX=  .*\. (c|cpp|h)
```

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 gzip'd 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. (The most common case is `EXTRACT_SUFX=.tar.Z`, when the tarball is condensed by regular `compress`, not `gzip`.)

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).

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 pre-empt 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 `gtkmmumble` port, version 0.10, is committed to the ports collection:

```
PORTNAME=      gtkmmumble
PORTVERSION=    0.10
```

`PKGNAME` becomes `gtkmmumble-0.10`.

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

```
PORTNAME=      gtkmmumble
PORTVERSION=    0.10
PORTREVISION=    1
```

`PKGNAME` becomes `gtkmmumble-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=      gtkmmumble
PORTVERSION=    0.2
PORTEPOCH=      1
```

`PKGNAME` becomes `gtkmmumble-0.2,1`

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

```
PORTNAME=      gtkmmumble
PORTVERSION=    0.3
PORTEPOCH=      1
```

`PKGNAME` becomes `gtkmmumble-0.3,1`

Note: If `PORTEPOCH` were reset to 0 with this upgrade, someone who had installed the `gtkmmumble-0.10_1` package would not detect the `gtkmmumble-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

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 `PKGNAME_SUFFIX`. In those cases, the optional `LATEST_LINK` variable should be set to a different value for all ports except the “main” one — see the `editors/vim5` and `editors/vim` ports, and the `www/apache*` family for examples of its use. 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}${PKGNAME_SUFFIX}-${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`, `PKGNAME_SUFFIX`, 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 `papersize` and `font units`.

The `-compiled.specifics` part should be set in the `PKGNAME_SUFFIX` variable.

5. The version string should follow a dash (`-`) and be a period-separated list of integers and single lowercase alphabets. In particular, it is not permissible to have another dash inside the version string. The only exception is the string `p1` (meaning “patchlevel”), 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 `yyyy.mm.dd` format, not `dd.mm.yyyy` or the non-Y2K compliant `yy.mm.dd` format.

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	PKGNAME_SUFFIX	PORTVERSION	Reason
mule-2.2.2	(empty)	mule	(empty)	2.2.2	No changes required
EmiClock-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?
tvtnm	(empty)	tvtnm	(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	Papersize hardcoded at package build time
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 (`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 (http://www.afterstep.org) window manager.	
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.	

Category	Description	Notes
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 <code>math</code>).
elisp*	Emacs-lisp ports.	
emulators	Emulators for other operating systems.	Terminal emulators do <i>not</i> belong here—X-based ones should go to <code>x11</code> and text-based ones to either <code>comms</code> or <code>misc</code> , depending on the exact functionality.
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 <code>ftp</code> with a secondary category of <code>www</code> .
games	Games.	
geography*	Geography-related software.	
german	German language support.	
gnome*	Ports from the GNOME (http://www.gnome.org) 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.	

Category	Description	Notes
java	Software related to the Java language.	The <code>java</code> category shall not be the only one for a port. Save for ports directly related to the Java language, porters are also encouraged not to use <code>java</code> as the main category of a port.
kde*	Ports from the K Desktop Environment (KDE) (http://www.kde.org) Project.	
kld*	Kernel loadable modules.	
korean	Korean language support.	
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 <code>misc</code> , 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™ (http://www.palm.com/) 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 (http://www.cs.bell-labs.com/plan9dist/).	
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 (http://www.python.org/) language.	
ruby*	Software related to the Ruby (http://www.ruby-lang.org/) language.	
rubygems*	Ports of RubyGems (http://www.rubygems.org/) packages.	
russian	Russian language support.	
scheme*	Software related to the Scheme language.	
science	Scientific ports that do not fit into other categories such as <code>astro</code> , <code>biology</code> and <code>math</code> .	
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 <code>print</code> .
tk*	Ports that use Tk to run.	
ukrainian	Ukrainian language support.	
vietnamese	Vietnamese language support.	
windowmaker*	Ports to support the WindowMaker window manager.	
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 <code>x11-*</code> categories (see below). If your port <i>is</i> an X application, define <code>USE_XLIB</code> (implied by <code>USE_IMAKE</code>) and put it in the appropriate category.
x11-clocks	X11 clocks.	
x11-drivers	X11 drivers.	
x11-fm	X11 file managers.	

Category	Description	Notes
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 (http://www.xfce.org/) desktop environment.	
zope*	Zope (http://www.zope.org/) 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`, `mbone`, `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.
- `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 (<http://lists.FreeBSD.org/mailman/listinfo/freebsd-ports>) 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 (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/committers-guide/#ports) 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 (<http://lists.FreeBSD.org/mailman/listinfo/freebsd-ports>). 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 (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/committers-guide/article.html#PORTS).

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_SUFFIX}`. 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:

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

Note: `PKGNAMEPREFIX` and `PKGNAME_SUFFIX` 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_SUFFIX}`, you should probably leave `DISTNAME` alone—you are better off defining `DISTFILES` than having to set both `DISTNAME` and `WRKSRC` (and possibly `EXTRACT_SUFFIX`).

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 X-contrib, GNU, or Perl CPAN, you may be able refer to those sites in an easy compact form using `MASTER_SITE_*` (e.g., `MASTER_SITE_XCONTRIB`,

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_XCONTRIB}
MASTER_SITE_SUBDIR= applications
```

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.

The user can also set the MASTER_SITE_* variables in `/etc/make.conf` to override our choices, and use their favorite mirrors of these popular archives instead.

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 and USE_ZIP variables automatically set EXTRACT_SUFX to `.tar.bz2` 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., `WRKSRCDIR`) because it contains some extra pathnames, set `PATCH_DIST_STRIP` accordingly. For instance, if all the pathnames in the patch have an extra `foozolib-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 gzip'd 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 gzip'd or compress'd 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` hereon.

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.

Example 5-1. Simplified use of `MASTER_SITES:n` with 1 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.

Example 5-2. Simplified use of `MASTER_SITES:n` with more than 1 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 postfixes 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 postfixes with `:n`. See both Example 5-3 and Example 5-4.

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.

Example 5-5. Detailed use of `MASTER_SITES` 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.

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.
- 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 iii.ii for more information on these new port targets.

iii. New port targets

- i. 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.
- ii. 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 sideeffects 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 (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/contributing-ports/maintain-port.html) 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* do not include the package name (or version number of the software) in the comment. The comment should begin with a capital and end without a period. Here is an example:

```
COMMENT=      A cat chasing a mouse all over the screen
```

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

Please try to keep the COMMENT line less than 70 characters, as it is displayed to users as a one-line summary of the port.

5.7 Dependencies

Many ports depend on other ports. There are seven variables that you can use 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.

5.7.1 LIB_DEPENDS

This variable specifies the shared libraries this port depends on. It is a list of *lib:dir[:target]* 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, and *target* is the target to call in that directory. For example,

```
LIB_DEPENDS=  jpeg.9:${PORTSDIR}/graphics/jpeg
```

will check for a shared jpeg library with major version 9, and descend into the `graphics/jpeg` subdirectory of your ports tree to build and install it if it is not found. The *target* part can be omitted if it is equal to `DEPENDS_TARGET` (which defaults to `install`).

Note: The *lib* part is a regular expression which is being looked up in the `ldconfig -r` output. Values such as `intl.[5-7]` and `intl` are allowed. The first pattern, `intl.[5-7]`, will match any of: `intl.5`, `intl.6` or `intl.7`. The second pattern, `intl`, will match any version of the `intl` library.

The dependency is checked twice, once from within the `extract` 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.

5.7.2 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=    ${LOCALBASE}/etc/innd:${PORTSDIR}/news/inn \
               xmlcatmgr:${PORTSDIR}/textproc/xmlcatmgr
```

will check if the file or directory `/usr/local/etc/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:/usr/X11R6/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`.

5.7.3 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`.

5.7.4 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`.

5.7.5 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 5.7.7.

5.7.6 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`.

5.7.7 USE_*

A number of variables exist in order to encapsulate common dependencies that many ports have. Although their use is optional, they can help to reduce the verbosity of the port Makefiles. Each of them is styled as `USE_*`. The usage

of these variables is restricted to the port Makefiles and `ports/Mk/bsd.*.mk` and is not designed to encapsulate user-settable options — use `WITH_*` and `WITHOUT_*` for that purpose.

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

```
USE_GCC=3.2
```

would add a dependency on `gcc32` for every port, including `gcc32` itself!

Table 5-1. The `USE_*` variables

Variable	Means
<code>USE_BZIP2</code>	The port's tarballs are compressed with <code>bzip2</code> .
<code>USE_ZIP</code>	The port's tarballs are compressed with <code>zip</code> .
<code>USE_BISON</code>	The port uses <code>bison</code> for building.
<code>USE_CDRTOOLS</code>	The port requires cdrecord either from <code>sysutils/cdrtools</code> or <code>sysutils/cdrtools-cjk</code> , according to the user's preference.
<code>USE_GCC</code>	The port requires a specific version of <code>gcc</code> to build. The exact version can be specified with value such as <code>3.2</code> . The minimal required version can be specified as <code>3.2+</code> . The <code>gcc</code> from the base system is used when it satisfies the requested version, otherwise an appropriate <code>gcc</code> is compiled from ports and the <code>CC</code> and <code>CXX</code> variables are adjusted.

Variables related to **gmake** and the `configure` script are described in Section 6.3, while **autoconf**, **automake** and **libtool** are described in Section 6.4. **Perl** related variables are described in Section 6.6. X11 variables are listed in Section 6.7. Section 6.8 deals with GNOME and Section 6.9 with KDE related variables. Section 6.10 documents Java variables, while Section 6.11 contains information on **Apache**, **PHP** and **PEAR** modules. **Python** is discussed in Section 6.12, while **Ruby** in Section 6.15. Section 6.16 provides variables used for **SDL** applications and finally, Section 6.19 contains information on **Xfce**.

5.7.8 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.

5.7.9 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.

5.7.10 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 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 `PKGNAME_SUFFIX` 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-
PKGNAME_SUFFIX=  ${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 `FILES_DIRS` and `SCRIPT_DIRS` 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.

5.9 Manpages

The `MAN[1-9LN]` variables will automatically add any manpages to `pkg-plist` (this means you must *not* list manpages in the `pkg-plist`—see generating `PLIST` for more). It also makes the install stage automatically compress or uncompress manpages depending on the setting of `NOMANCOMPRESS` in `/etc/make.conf`.

If your port tries to install multiple names for manpages using symlinks or hardlinks, you must use the `MLINKS` variable to identify these. The link installed by your port will be destroyed and recreated by `bsd.port.mk` to make sure it points to the correct file. Any manpages listed in `MLINKS` must not be listed in the `pkg-plist`.

To specify whether the manpages are compressed upon installation, use the `MANCOMPRESSED` variable. This variable can take three values, `yes`, `no` and `maybe`. `yes` means manpages are already installed compressed, `no` means they are not, and `maybe` means the software already respects the value of `NOMANCOMPRESS` so `bsd.port.mk` does not have to do anything special.

`MANCOMPRESSED` is automatically set to `yes` if `USE_IMAKE` is set and `NO_INSTALL_MANPAGES` is not set, and to `no` otherwise. You do not have to explicitly define it unless the default is not suitable for your port.

If your port anchors its man tree somewhere other than `MANPREFIX`, you can use the `MANPREFIX` to set it. Also, if only manpages in certain sections go in a non-standard place, such as some `perl` modules ports, you can set individual man paths using `MANsectPREFIX` (where *sect* is one of 1-9, L or N).

If your manpages go to language-specific subdirectories, set the name of the languages to `MANLANG`. The value of this variable defaults to `"` (i.e., English only).

Here is an example that puts it all together.

```

MAN1=             foo.1
MAN3=             bar.3
MAN4=             baz.4
MLINKS=           foo.1 alt-name.8
MANLANG=          "" ja
MAN3PREFIX=       ${PREFIX}/share/foobar

```

```
MANCOMPRESSED= yes
```

This states that six files are installed by this port;

```
{MANPREFIX}/man/man1/foo.1.gz
{MANPREFIX}/man/ja/man1/foo.1.gz
{PREFIX}/share/foobar/man/man3/bar.3.gz
{PREFIX}/share/foobar/man/ja/man3/bar.3.gz
{MANPREFIX}/man/man4/baz.4.gz
{MANPREFIX}/man/ja/man4/baz.4.gz
```

Additionally `{MANPREFIX}/man/man8/alt-name.8.gz` may or may not be installed by your port. Regardless, a symlink will be made to join the `foo(1)` manpage and `alt-name(8)` manpage.

If only some manpages are translated, you can use several variables dynamically created from `MANLANG` content:

```
MANLANG=      "" de ja
MAN1=         foo.1
MAN1_EN=      bar.1
MAN3_DE=      baz.3
```

This translates into this list of files:

```
{MANPREFIX}/man/man1/foo.1.gz
{MANPREFIX}/man/de/man1/foo.1.gz
{MANPREFIX}/man/ja/man1/foo.1.gz
{MANPREFIX}/man/man1/bar.1.gz
{MANPREFIX}/man/de/man3/baz.3.gz
```

5.10 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/gcc33` installs info files to `PREFIX/INFO_PATH/gcc33`, and `INFO` will be something like this:

```
INFO= gcc33/cpp gcc33/cppinternals gcc33/g77 ...
```

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

5.11 Makefile Options

Some large applications can be built in a number of configurations, adding functionality if one of a number of libraries or applications is available. Examples include choice of natural (human) language, GUI versus command-line, or type of database to support. Since not all users want those libraries or applications, the ports system provides hooks that the port author can use to control which configuration should be built. Supporting these properly will make users happy, and effectively provide 2 or more ports for the price of one.

5.11.1 Knobs

5.11.1.1 `WITH_*` and `WITHOUT_*`

These variables are designed to be set by the system administrator. There are many that are standardized in `ports/KNOBS` (<http://www.freebsd.org/cgi/cvsweb.cgi/ports/KNOBS?rev=HEAD&content-type=text/x-cvsweb-markup>) 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 some kind of variable such as `YES` or `NO`.

Table 5-2. Common `WITH_*` and `WITHOUT_*` variables

Variable	Means
<code>WITHOUT-NLS</code>	If set, says that internationalization is not needed, which can save compile time. By default, internationalization is used.
<code>WITH_OPENSSL_BASE</code>	Use the version of OpenSSL in the base system.
<code>WITH_OPENSSL_PORT</code>	Installs the version of OpenSSL from <code>security/openssl</code> , even if the base is up to date.
<code>WITHOUT_X11</code>	If the port can be built both with and without X support, then it should normally be built with X support. If this variable is defined, then the version that does not have X support should be built instead.

5.11.1.2 Knob naming

It is recommended that porters use like-named knobs, 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` (<http://www.freebsd.org/cgi/cvsweb.cgi/ports/KNOBS?rev=HEAD&content-type=text/x-cvsweb-markup>) 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.

5.11.2 OPTIONS

5.11.2.1 Background

The `OPTIONS` variable gives the user who installs the port a dialog with the available options and saves them to `/var/db/ports/portname/options`. Next time when the port has to be rebuilt, the options are reused. Never again you will have to remember all the twenty `WITH_*` and `WITHOUT_*` options you used to build this port!

When the user runs `make config` (or runs `make build` for the first time), the framework will check for `/var/db/ports/portname/options`. If that file does not exist, it will use the values of `OPTIONS` to create a dialogbox where the options can be enabled or disabled. Then the `options` file is saved and the selected variables will be 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.

Use `make showconfig` to see the saved configuration. Use `make rmconfig` to remove the saved configuration.

5.11.2.2 Syntax

The syntax for the `OPTIONS` variable is:

```
OPTIONS= OPTION "descriptive text" default ...
```

The value for default is either `ON` or `OFF`. Multiple repetitions of these three fields are allowed.

`OPTIONS` definition must appear before the inclusion of `bsd.port.options.mk`. The `WITH_*` and `WITHOUT_*` variables 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 `USE_*` flags.

Example 5-8. Simple use of `OPTIONS`

```
OPTIONS=      FOO "Enable option foo" On \
              BAR "Support feature bar" Off

.include <bsd.port.options.mk>

.if defined(WITHOUT_FOO)
CONFIGURE_ARGS+= --without-foo
.else
CONFIGURE_ARGS+= --with-foo
.endif

.if defined(WITH_BAR)
RUN_DEPENDS+= bar:${PORTSDIR}/bar/bar
.endif

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

Example 5-9. Old style use of `OPTIONS`

```

OPTIONS=      FOO "Enable option foo" On

.include <bsd.port.pre.mk>

.if defined(WITHOUT_FOO)
CONFIGURE_ARGS+= --without-foo
.else
CONFIGURE_ARGS+= --with-foo
.endif

.include <bsd.port.post.mk>

```

5.11.3 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 5-10. Wrong handling of an option

```

.if defined(WITH_FOO)
LIB_DEPENDS+= foo.0:${PORTSDIR}/devel/foo
CONFIGURE_ARGS+= --enable-foo
.endif

```

In the example above, imagine a library `libfoo` is installed on the system. 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 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 5-11. Correct handling of an option

```

.if defined(WITH_FOO)
LIB_DEPENDS+= foo.0:${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.

5.12 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.

5.12.1 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}
```

5.12.2 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
```

5.13 CONFLICTS

If your package cannot coexist with other packages (because of file conflicts, runtime incompatibility, etc.), list the other package names 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` does not match this port's package itself, or else forcing its installation with `FORCE_PKG_REGISTER` will no longer work.

Note: `CONFLICTS` automatically sets `IGNORE`, which is more fully documented in Section 12.15.

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 Installing files

5.14.1 INSTALL_* macros

Do use the macros provided in `bsd.port.mk` to ensure correct modes and ownership of files in your own `*-install` targets.

- `INSTALL_PROGRAM` is a command to install binary executables.
- `INSTALL_SCRIPT` is a command to install executable scripts.
- `INSTALL_KLD` is a command to install kernel loadable modules. Some architectures don't like it when the modules are stripped, therefore 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.

5.14.2 Stripping Binaries

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).

If you need to strip a file, but do not wish to use the `INSTALL_PROGRAM` macro, `${STRIP_CMD}` will strip your program. This is typically done within the `post-install` target. For example:

```
post-install:
${STRIP_CMD} ${PREFIX}/bin/xd1
```

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.

5.14.3 Installing a whole tree of files

Sometimes, there is a need to install a big number of files, preserving their hierarchical organization, ie. copying over a whole directory tree from `WRKSRC` to a target directory under `PREFIX`.

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} ${EXAMPLESDIR}
(cd ${WRKSRC}/examples/ && ${COPYTREE_SHARE} \* ${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} ${DATADIR}/summer
(cd ${WRKSRCDIR}/temperatures/ && ${COPYTREE_SHARE} "June July August" ${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} ${EXAMPLESDIR}
(cd ${WRKSRCDIR}/examples/ && \
${COPYTREE_SHARE} \* ${EXAMPLESDIR} "! -name Makefile")
```

Note that these macros does not add the installed files to `pkg-plist`. You still need to list them.

5.14.4 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`.

Make the installation dependent on the variable `NOPORTDOCS` so that users can disable it in `/etc/make.conf`, like this:

```
post-install:
.if !defined(NOPORTDOCS)
${MKDIR} ${DOCSDIR}
${INSTALL_MAN} ${WRKSRCDIR}/docs/xvdocs.ps ${DOCSDIR}
.endif
```

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: `NOPORTDOCS` 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 `NOPORTDATA` and `NOPORTEXAMPLES`, 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 `NOPORTDOCS` is defined then files and directories listed in `PORTDOCS` would not be installed and neither would be 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 `PORTEXTAMPLES`, 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`.

5.14.5 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 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/mozilla` 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.

6.2 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 (<http://lists.FreeBSD.org/mailman/listinfo/freebsd-ports>).

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

6.2.1 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.

6.2.2 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.

6.2.3 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.

6.2.4 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.

6.2.5 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 Building mechanisms

6.3.1 Parallel ports building

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.

6.3.2 `make`, `gmake`, and `imake`

If your port uses **GNU make**, set `USE_GMAKE=yes`.

Table 6-1. Variables for ports related to `gmake`

Variable	Means
<code>USE_GMAKE</code>	The port requires <code>gmake</code> to build.
<code>GMAKE</code>	The full path for <code>gmake</code> if it is not in the <code>PATH</code> .

If your port is an X application that creates `Makefile` files from `Imakefile` files using **`imake`**, then set `USE_IMAKE=yes`. 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`.

6.3.3 `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 <code>configure</code> 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 <code>configure</code> script.
CONFIGURE_ENV	Additional environment variables to be set for <code>configure</code> script run.
CONFIGURE_TARGET	Override default configure target. Default value is <code>\${MACHINE_ARCH}-portbld-freebsd\${OSREL}</code> .

6.3.4 Using `scons`

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

Table 6-3. 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 <code>MAKE_TARGET</code> .

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`.

6.4 Using GNU autotools

6.4.1 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.

6.4.2 `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.

6.4.3 `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.

6.4.4 `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.

6.4.5 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.

6.5 Using GNU gettext

6.5.1 Basic usage

If your port requires `gettext`, just set `USE_GETTEXT` to `yes`, and your port will grow the dependency on `devel/gettext`. The value of `USE_GETTEXT` can also specify the required version of the `libintl` library, the basic part of `gettext`, but using this feature is *strongly discouraged*: Your port should work with just the current version of `devel/gettext`.

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:

```
USE_GETTEXT=    yes
CPPFLAGS+=      -I${LOCALBASE}/include
LDFlags+=       -L${LOCALBASE}/lib

GNU_CONFIGURE=  yes
CONFIGURE_ENV=  CPPFLAGS="${CPPFLAGS}" \
                LDFlags="${LDFlags}"
```

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

```
USE_GETTEXT=      yes
GNU_CONFIGURE=    yes
CONFIGURE_ENV=    CPPFLAGS="-I${LOCALBASE}/include" \
                  LDFLAGS="-L${LOCALBASE}/lib"
```

6.5.2 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 `WITHOUT-NLS`. For ports of low to medium complexity, you can rely on the following idiom:

```
GNU_CONFIGURE=      yes

.if !defined(WITHOUT-NLS)
USE_GETTEXT=        yes
PLIST_SUB+=         NLS=" "
.else
CONFIGURE_ARGS+=    --disable-nls
PLIST_SUB+=         NLS="@comment "
.endif
```

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.

6.5.3 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 your port. The most popular languages have their respective directories listed in `/etc/mtree/BSD.local.dist`; that is, they are a part of the base system. The directories for many other languages are governed by the `devel/gettext` port. You may want to consult its `pkg-plist` and see whether your port is going to install a message catalog file for a unique language.

6.6 Using perl

If `MASTER_SITES` is set to `MASTER_SITE_PERL_CPAN`, then preferred value of `MASTER_SITE_SUBDIR` is top-level hierarchy name. For example, the recommend value for `p5-Module-Name` is `Module`. The top-level hierarchy can be examined at `cpan.org` (<http://cpan.org/modules/by-module/>). 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 the directory. In such case, using author's id as `MASTER_SITE_SUBDIR` is allowed.

All of the tunable knobs below accept both `YES` and a version string, like `5.8.0+`. Using `YES` means that the port can be used with all 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 minimal version (e.g. `5.7.3+`), a maximal version (e.g. `5.8.0-`) or an exact version (e.g. `5.8.3`).

Table 6-4. Variables for ports that use perl

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

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

6.7 Using X11

6.7.1 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 evieproto fixesproto
fontcacheproto fontenc fontspROTO fontutil glproto ice inputproto kbproto libfs oldx
printproto randrproto recordproto renderproto resourceproto scrnsaverproto sm
trapproto videoproto x11 xau xaw xaw6 xaw7 xaw8 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 xpr
oto 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`, `gl` and `linux`. For backwards compatibility, the value of `yes` maps to `glu`.

Example 6-1. `USE_XORG` example

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

Many ports define `USE_XLIB`, which makes the port depend on all the 50 or so libraries. This variable exists for backwards compatibility, as it predates modular X.Org, and should not be used on new ports.

Table 6-5. Variables for ports that use X

<code>USE_XLIB</code>	The port uses the X libraries. Deprecated - use a list of X.Org components in <code>USE_XORG</code> variable instead.
<code>USE_IMAKE</code>	The port uses <code>imake</code> .
<code>USE_X_PREFIX</code>	Deprecated. Today it is equivalent to <code>USE_XLIB</code> and can be replaced by it freely.
<code>XMKMF</code>	Set to the path of <code>xmkmf</code> if not in the <code>PATH</code> . Defaults to <code>xmkmf -a</code> .

Table 6-6. Variables for depending on individual parts of X11

<code>X_IMAKE_PORT</code>	Port providing <code>imake</code> and several other utilities used to build X11.
<code>X_LIBRARIES_PORT</code>	Port providing X11 libraries.
<code>X_CLIENTS_PORT</code>	Port providing X clients.
<code>X_SERVER_PORT</code>	Port providing X server.

<code>X_FONTSERVER_PORT</code>	Port providing font server.
<code>X_PRINTSERVER_PORT</code>	Port providing print server.
<code>X_VFBSERVER_PORT</code>	Port providing virtual framebuffer server.
<code>X_NESTSERVER_PORT</code>	Port providing a nested X server.
<code>X_FONTS_ENCODINGS_PORT</code>	Port providing encodings for fonts.
<code>X_FONTS_MISC_PORT</code>	Port providing miscellaneous bitmap fonts.
<code>X_FONTS_100DPI_PORT</code>	Port providing 100dpi bitmap fonts.
<code>X_FONTS_75DPI_PORT</code>	Port providing 75dpi bitmap fonts.
<code>X_FONTS_CYRILLIC_PORT</code>	Port providing cyrillic bitmap fonts.
<code>X_FONTS_TTF_PORT</code>	Port providing TrueType® fonts.
<code>X_FONTS_TYPE1_PORT</code>	Port providing Type1 fonts.
<code>X_MANUALS_PORT</code>	Port providing developer oriented manual pages

Example 6-2. Using X11 related variables in port

```
# Use some X11 libraries and depend on
# font server as well as cyrillic fonts.
RUN_DEPENDS=    ${LOCALBASE}/bin/xfs:${X_FONTSERVER_PORT} \
                ${LOCALBASE}/lib/X11/fonts/cyrillic/crox1c.pcf.gz:${X_FONTS_CYRILLIC_PORT}

USE_XORG=       x11 xpm
```

6.7.2 Ports that require Motif

If your port requires a Motif library, define `USE_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/X11R6/lib -lXm` or `/usr/X11R6/lib/libXm.a`, so there is no need to add `-L` or `-l` in front.

6.7.3 X11 fonts

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

6.7.4 Getting fake `DISPLAY` using `Xvfb`

Some applications require a working X11 display for compilation to succeed. This poses 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.

```
USE_DISPLAY= yes
```

6.7.5 Desktop entries

Desktop Entries (Freedesktop standard (<http://standards.freedesktop.org/desktop-entry-spec/latest/>)) can be easily created in your port using `DESKTOP_ENTRIES` variable. These entries do show up in application menus of compliant desktop environments like GNOME or KDE. The `.desktop` file 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 (<http://standards.freedesktop.org/menu-spec/latest/apa.html>). The `StartupNotify` indicates, if the application will clear the status in startup notification aware environment.

Example:

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

6.8 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 (<http://www.FreeBSD.org/gnome/docs/porting.html>) exists within the FreeBSD/GNOME project's homepage.

6.9 Using KDE

6.9.1 Variable definitions

Table 6-7. Variables for ports that use KDE

`USE_KDELIBS_VER`

The port uses KDE libraries. It specifies the major version of KDE to use and implies `USE_QT_VER` of the appropriate version. The only possible value is 3.

USE_KDEBASE_VER

The port uses KDE base. It specifies the major version of KDE to use and implies `USE_QT_VER` of the appropriate version. The only possible value is 3.

6.9.2 Ports that require Qt

Table 6-8. Variables for ports that use Qt

USE_QT_VER	The port uses the Qt toolkit. Possible values are 3 and 4; each specify the major version of Qt to use. Appropriate parameters are passed to <code>configure</code> script and <code>make</code> .
QT_PREFIX	Set to the path where Qt installed to (read-only variable).
MOC	Set to the path of <code>moc</code> (read-only variable). Default set according to <code>USE_QT_VER</code> value.
QTCPPFLAGS	Additional compiler flags passed via <code>CONFIGURE_ENV</code> for Qt toolkit. Default set according to <code>USE_QT_VER</code> .
QTCFGLIBS	Additional libraries for linking passed via <code>CONFIGURE_ENV</code> for Qt toolkit. Default set according to <code>USE_QT_VER</code> .
QTNONSTANDARD	Suppress modification of <code>CONFIGURE_ENV</code> , <code>CONFIGURE_ARGS</code> , and <code>MAKE_ENV</code> .

Table 6-9. Additional variables for ports that use Qt 4.x

QT_COMPONENTS	Specify tool and library dependencies for Qt4. See below for details.
UIC	Set to the path of <code>uic</code> (read-only variable). Default set according to <code>USE_QT_VER</code> value.
QMAKE	Set to the path of <code>qmake</code> (read-only variable). Default set according to <code>USE_QT_VER</code> value.
QMAKESPEC	Set to the path of configuration file for <code>qmake</code> (read-only variable). Default set according to <code>USE_QT_VER</code> value.

When `USE_QT_VER` is set, some useful settings are passed to `configure` script:

```
CONFIGURE_ARGS+= --with-qt-includes=${QT_PREFIX}/include \
                 --with-qt-libraries=${QT_PREFIX}/lib \
                 --with-extra-libs=${LOCALBASE}/lib \
                 --with-extra-includes=${LOCALBASE}/include
CONFIGURE_ENV+= MOC="${MOC}" CPPFLAGS="${CPPFLAGS} ${QTCPPFLAGS}" LIBS="${QTCFGLIBS}" \
                QTDIR="${QT_PREFIX}" KDEDIR="${KDE_PREFIX}"
```

If `USE_QT_VER` is set to 4, the following settings are also deployed:


```
CONFIGURE_ENV+= UIC="${UIC}" QMAKE="${QMAKE}" QMAKESPEC="${QMAKESPEC}"
MAKE_ENV+=      QMAKESPEC="${QMAKESPEC}"
```

6.9.3 Component selection (Qt 4.x only)

When `USE_QT_VER` is set to 4, individual Qt4 tool and library dependencies can be specified in the `QT_COMPONENTS` 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 `_QT_COMPONENTS_ALL` in `/usr/ports/Mk/bsd.qt.mk`):

Table 6-10. Available Qt4 library components

Name	Description
<code>corelib</code>	core library (can be omitted unless the port uses nothing but <code>corelib</code>)
<code>gui</code>	graphical user interface library
<code>network</code>	network library
<code>opengl</code>	OpenGL library
<code>qt3support</code>	Qt3 compatibility library
<code>qtestlib</code>	unit testing library
<code>script</code>	script library
<code>sql</code>	SQL library
<code>xml</code>	XML library

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

Table 6-11. Available Qt4 tool components

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

Table 6-12. Available Qt4 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 6-3. Selecting Qt4 components

In this example, the ported application uses the Qt4 graphical user interface library, the Qt4 core library, all of the Qt4 code generation tools and Qt4's Makefile generator. Since the gui library implies a dependency on the core library, corelib does not need to be specified. The Qt4 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_QT_VER=      4
QT_COMPONENTS=  gui moc_build qmake_build rcc_build uic_build
```

6.9.4 Additional considerations

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

```
HAS_CONFIGURE= yes
```

```
do-configure:
    @cd ${WRKSRCDIR} && ${SETENV} ${CONFIGURE_ENV} \
        ${QMAKE} -unix PREFIX=${PREFIX} texmaker.pro
```

Note the similarity to the `qmake` line from the provided `BUILD.sh` script. Passing `CONFIGURE_ENV` ensures `qmake` will see the `QMAKESPEC` variable, without which it cannot work. `qmake` generates standard Makefiles, so it is not necessary to write our own build target.

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

- *Missing additional includepaths.* 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 searchpaths via the commandline, for example:

```
${QMAKE} -unix PREFIX=${PREFIX} INCLUDEPATH+=${LOCALBASE}/include \
LIBS+=-L${LOCALBASE}/lib sillyapp.pro
```

- *Bogus installation paths.* Sometimes data such as icons or `.desktop` files are by default installed into directories which aren't scanned by XDG-compatible applications. `editors/texmaker` is an example for this - look at `patch-texmaker.pro` in the `files` directory of that port for a template on how to remedy this directly in the Qmake project file.

6.10 Using Java

6.10.1 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 is `java/jdk15`.

Table 6-13. Variables that may be set by ports that use Java

Variable	Means
<code>USE_JAVA</code>	Should be defined for the remaining variables to have any effect.
<code>JAVA_VERSION</code>	List of space-separated suitable Java versions for the port. An optional "+" allows you to specify a range of versions (allowed values: <code>1.1[+]</code> <code>1.2[+]</code> <code>1.3[+]</code> <code>1.4[+]</code>).
<code>JAVA_OS</code>	List of space-separated suitable JDK port operating systems for the port (allowed values: <code>native</code> <code>linux</code>).
<code>JAVA_VENDOR</code>	List of space-separated suitable JDK port vendors for the port (allowed values: <code>freebsd</code> <code>bsdjava</code> <code>sun</code> <code>ibm</code> <code>blackdown</code>).
<code>JAVA_BUILD</code>	When set, it means that the selected JDK port should be added to the build dependencies of the port.
<code>JAVA_RUN</code>	When set, it means that the selected JDK port should be added to the run dependencies of the port.
<code>JAVA_EXTRACT</code>	When set, it means that the selected JDK port should be added to the extract dependencies of the port.
<code>USE_JIKES</code>	Whether the port should or should not use the <code>jikes</code> bytecode compiler to build. When no value is set for this variable, the port will use <code>jikes</code> to build if available. You may also explicitly forbid or enforce the use of <code>jikes</code> (by setting <code>'no'</code> or <code>'yes'</code>). In the later case, <code>devel/jikes</code> will be added to build dependencies of the port. In any case that <code>jikes</code> is actually used in place of <code>javac</code> , then the <code>HAVE_JIKES</code> variable is defined by <code>bsd.java.mk</code> .

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

Table 6-14. Variables provided to ports that use Java

Variable	Value
<code>JAVA_PORT</code>	The name of the JDK port (e.g. <code>'java/jdk14'</code>).

Variable

JAVA_PORT_VERSION

JAVA_PORT_OS

JAVA_PORT_VENDOR

JAVA_PORT_OS_DESCRIPTION

JAVA_PORT_VENDOR_DESCRIPTION

JAVA_HOME

JAVAC

JAR

APPLETVIEWER

JAVA

JAVADOC

JAVAH

JAVAP

JAVA_KEYTOOL

JAVA_N2A

JAVA_POLICYTOOL

JAVA_SERIALVER

RMIC

RMIREGISTRY

RMID

JAVA_CLASSES

HAVE_JIKES

Value

The full version of the JDK port (e.g. '1.4.2'). If you only need the first two digits of this version number, use `${JAVA_PORT_VERSION:C/^[0-9])\.([0-9])(.*)$/\1.\2/}`.

The operating system used by the JDK port (e.g. 'linux').

The vendor of the JDK port (e.g. 'sun').

Description of the operating system used by the JDK port (e.g. 'Linux').

Description of the vendor of the JDK port (e.g. 'FreeBSD Foundation').

Path to the installation directory of the JDK (e.g. '/usr/local/jdk1.3.1').

Path to the Java compiler to use (e.g. '/usr/local/jdk1.1.8/bin/javac' or '/usr/local/bin/jikes').

Path to the jar tool to use (e.g. '/usr/local/jdk1.2.2/bin/jar' or '/usr/local/bin/fastjar').

Path to the appletviewer utility (e.g. '/usr/local/linux-jdk1.2.2/bin/appletviewer').

Path to the java executable. Use this for executing Java programs (e.g. '/usr/local/jdk1.3.1/bin/java').

Path to the javadoc utility program.

Path to the javah program.

Path to the javap program.

Path to the keytool utility program. This variable is available only if the JDK is Java 1.2 or higher.

Path to the native2ascii tool.

Path to the policytool program. This variable is available only if the JDK is Java 1.2 or higher.

Path to the serialver utility program.

Path to the RMI stub/skeleton generator, rmic.

Path to the RMI registry program, rmiregistry.

Path to the RMI daemon program rmid. This variable is only available if the JDK is Java 1.2 or higher.

Path to the archive that contains the JDK class files. On JDK 1.2 or later, this is

`${JAVA_HOME}/jre/lib/rt.jar`. Earlier JDKs used `${JAVA_HOME}/lib/classes.zip`.

Defined whenever jikes is used by the port (see `USE_JIKES` above).

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 6-15. Constants defined for ports that use Java

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

The related entries are defined in both `PLIST_SUB` (documented in Section 7.1) and `SUB_LIST`.

6.10.2 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_TARGETS`. This is similar to the `USE_GMAKE` mechanism, which is documented in Section 6.3.

If `jikes` is used in place of `javac` (see `USE_JIKES` in Section 6.10.1), then Ant will automatically use it to build the port.

6.10.3 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 to 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) 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 (<http://www.freebsd.org/java/>) 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` CVS log (<http://www.freebsd.org/cgi/cvsweb.cgi/ports/Mk/bsd.java.mk>) 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 (<http://lists.FreeBSD.org/mailman/listinfo/freebsd-java>).

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.

6.11 Web applications, Apache and PHP

6.11.1 Apache

Table 6-16. Variables for ports that use Apache

<code>USE_APACHE</code>	The port requires Apache. Possible values: <code>yes</code> (gets any version), <code>1.3</code> , <code>2.0</code> , <code>2.2</code> , <code>2.0+</code> , etc. Default dependency is on version <code>1.3</code> .
<code>WITH_APACHE2</code>	The port requires Apache 2.0. Without this variable, the port will depend on Apache 1.3. This variable is deprecated and should not be used anymore.
<code>APXS</code>	Full path to the <code>apxs</code> binary. Can be overridden in your port.
<code>HTTPD</code>	Full path to the <code>httpd</code> binary. Can be overridden in your port.
<code>APACHE_VERSION</code>	The version of present Apache installation (read-only variable). This variable is only available after inclusion of <code>bsd.port.pre.mk</code> . Possible values: <code>13</code> , <code>20</code> , <code>22</code> .
<code>APACHEMODDIR</code>	Directory for Apache modules. This variable is automatically expanded in <code>pkg-plist</code> .
<code>APACHEINCLUDEDIR</code>	Directory for Apache headers. This variable is automatically expanded in <code>pkg-plist</code> .
<code>APACHEETCDIR</code>	Directory for Apache configuration files. This variable is automatically expanded in <code>pkg-plist</code> .

Table 6-17. Useful variables for porting Apache modules

MODULENAME	Name of the module. Default value is <code>PORTNAME</code> . Example: <code>mod_hello</code>
SHORTMODNAME	Short name of the module. Automatically derived from <code>MODULENAME</code> , but can be overridden. Example: <code>hello</code>
AP_FAST_BUILD	Use <code>apxs</code> to compile and install the module.
AP_GENPLIST	Also automatically creates a <code>pkg-plist</code> .
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 <code>apxs</code> .

6.11.2 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.

6.11.3 PHP

Table 6-18. Variables for ports that use PHP

USE_PHP	The port requires PHP. The value <code>yes</code> adds a dependency on PHP. The list of required PHP extensions can be specified instead. Example: <code>pcre xml gettext</code>
DEFAULT_PHP_VER	Selects which major version of PHP will be installed as a dependency when no PHP is installed yet. Default is 4. 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.

6.11.4 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 6-4. 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>

```

6.12 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 6-19. Most useful variables for ports that use Python

<code>USE_PYTHON</code>	The port needs Python. Minimal required version can be specified with values such as <code>2.3+</code> . Version ranges can also be specified, by separating two version numbers with a dash, e.g.: <code>2.1-2.3</code>
<code>USE_PYDISTUTILS</code>	Use Python distutils for configuring, compiling and installing. This is required when the port comes with <code>setup.py</code> . This overrides the <code>do-build</code> and <code>do-install</code> targets and may also override <code>do-configure</code> if <code>GNU_CONFIGURE</code> is not defined.
<code>PYTHON_PKGNAMEPREFIX</code>	Used as a <code>PKGNAMEPREFIX</code> to distinguish packages for different Python versions. Example: <code>py24-</code>
<code>PYTHON_SITELIBDIR</code>	Location of the site-packages tree, that contains installation path of Python (usually <code>LOCALBASE</code>). The <code>PYTHON_SITELIBDIR</code> variable can be very useful when installing Python modules.
<code>PYTHONPREFIX_SITELIBDIR</code>	The PREFIX-clean variant of <code>PYTHON_SITELIBDIR</code> . Always use <code>%%PYTHON_SITELIBDIR%%</code> in <code>pkg-plist</code> when possible. The default value of <code>%%PYTHON_SITELIBDIR%%</code> is <code>lib/python%%PYTHON_VERSION%%/site-packages</code>
<code>PYTHON_CMD</code>	Python interpreter command line, including version number.
<code>PYNUMERIC</code>	Dependency line for numeric extension.
<code>PYNUMPY</code>	Dependency line for the new numeric extension, numpy. (<code>PYNUMERIC</code> is deprecated by upstream vendor).
<code>PYXML</code>	Dependency line for XML extension (not needed for Python 2.0 and higher as it is also in base distribution).
<code>USE_TWISTED</code>	Add dependency on twistedCore. The list of required components can be specified as a value of this variable. Example: <code>web lore pair flow</code>
<code>USE_ZOPE</code>	Add dependency on Zope, a web application platform. Change Python dependency to Python 2.3. Set <code>ZOPEBASEDIR</code> containing a directory with Zope installation.

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

6.13 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 6-20. The most useful variables for ports that use Tcl/Tk

<code>USE_TCL</code>	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 <code>INVALID_TCL_VER</code> variable.
<code>USE_TCL_BUILD</code>	The port needs Tcl only during the build time.
<code>USE_TCL_WRAPPER</code>	Ports that require the Tcl shell and do not require a specific <code>tclsh</code> version should use this new variable. The <code>tclsh</code> wrapper is installed on the system. The user can specify the desired <code>tcl</code> shell to use.
<code>WITH_TCL_VER</code>	User-defined variable that sets the desired Tcl version.
<code>UNIQUENAME_WITH_TCL_VER</code>	Like <code>WITH_TCL_VER</code> , but per-port.
<code>USE_TCL_THREADS</code>	Require a threaded build of Tcl/Tk .
<code>USE_TK</code>	The port depends on the Tk library (not the wish shell). Implies <code>USE_TCL</code> with the same value. For more information see the description of <code>USE_TCL</code> variable.
<code>USE_TK_BUILD</code>	Analog to the <code>USE_TCL_BUILD</code> variable.
<code>USE_TK_WRAPPER</code>	Analog to the <code>USE_TCL_WRAPPER</code> variable.
<code>WITH_TK_VER</code>	Analog to the <code>WITH_TCL_VER</code> variable and implies <code>WITH_TCL_VER</code> of the same value.

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

6.14 Using Emacs

This section is yet to be written.

6.15 Using Ruby

Table 6-21. Useful variables for ports that use Ruby

Variable	Description
<code>USE_RUBY</code>	The port requires Ruby.
<code>USE_RUBY_EXTCONF</code>	The port uses <code>extconf.rb</code> to configure.
<code>USE_RUBY_SETUP</code>	The port uses <code>setup.rb</code> to configure.

Variable	Description
RUBY_SETUP	Set to the alternative name of <code>setup.rb</code> . Common value is <code>install.rb</code> .

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 6-22. Selected read-only variables for ports that use Ruby

Variable	Description	Example value
RUBY_PKGNAMEPREFIX	Used as a <code>PKGNAMEPREFIX</code> to distinguish packages for different Ruby versions.	<code>ruby18-</code>
RUBY_VERSION	Full version of Ruby in the form of <code>x.y.z</code> .	<code>1.8.2</code>
RUBY_SITELIBDIR	Architecture independent libraries installation path.	<code>/usr/local/lib/ruby/site_ruby/1.8</code>
RUBY_SITEARCHLIBDIR	Architecture dependent libraries installation path.	<code>/usr/local/lib/ruby/site_ruby/1.8/amd64-fr</code>
RUBY_MOODOCDIR	Module documentation installation path.	<code>/usr/local/share/doc/ruby18/patsy</code>
RUBY_MODEXAMPLESDIR	Module examples installation path.	<code>/usr/local/share/examples/ruby18/patsy</code>

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

6.16 Using SDL

The `USE_SDL` variable is used to autoconfigure the dependencies for ports which use an SDL based library like `devel/sdl12` and `x11-toolkits/sdl_gui`.

The following SDL libraries are recognized at the moment:

- **sdl:** `devel/sdl12`
- **gfx:** `graphics/sdl_gfx`
- **gui:** `x11-toolkits/sdl_gui`
- **image:** `graphics/sdl_image`
- **ldbad:** `devel/sdl_ldbad`
- **mixer:** `audio/sdl_mixer`
- **mm:** `devel/sdlmm`
- **net:** `net/sdl_net`
- **sound:** `audio/sdl_sound`

- ttf: graphics/sdl_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`, 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`

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>
```

6.17 Using wxWidgets

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

6.17.1 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.

6.17.2 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 6-23. 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 6-24. 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 6.17.4).

The variables in Table 6-23 can be set to one or more of the following combinations separated by spaces:

Table 6-25. wxWidgets version specifications

Description	Example
Single version	2.4
Ascending range	2.4+
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 6-26. Variables to select preferred wxWidgets versions

Name	Designed for
WANT_WX_VER	the port
WITH_WX_VER	the user

6.17.3 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 6-27. Available wxWidgets components

Name	Description	Version restriction
------	-------------	---------------------

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 6-29). The following types are available:

Table 6-28. 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 6-29. Default wxWidgets dependency types

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

Example 6-5. 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
```

6.17.4 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 6-30. Variables to select Unicode in wxWidgets versions

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

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.

6.17.5 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 6-6. 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) || ${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) || ${HAVE_WX:Mpython} != ""
WX_COMPS+=        python
CONFIGURE_ARGS+=--enable-wxpython
.endif
```

6.17.6 Defined variables

The following variables are available in the port (after defining one from Table 6-23).

Table 6-31. Variables defined for ports that use wxWidgets

Name	Description
WX_CONFIG	The path to the wxWidgets <code>wx-config</code> script (with different name)
WXRC_CMD	The path to the wxWidgets <code>wxrc</code> 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

6.17.7 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 6-7. 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`.

6.17.8 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 6-32. Legal values for `WX_CONF_ARGS`

Possible value	Resulting argument
absolute	<code>--with-wx-config=\${WX_CONFIG}</code>
relative	<code>--with-wx=\${LOCALBASE}</code> <code>--with-wx-config=\${WX_CONFIG:T}</code>

6.18 Using Lua

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

6.18.1 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.

6.18.2 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 6-33. Variables to select Lua versions

Variable	Description	Default value
<code>USE_LUA</code>	List of versions the port can use	All available versions
<code>USE_LUA_NOT</code>	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 6-34. Available Lua versions

Version	Port
4.0	<code>lang/lua4</code>
5.0	<code>lang/lua50</code>
5.1	<code>lang/lua</code>

The variables in Table 6-33 can be set to one or more of the following combinations separated by spaces:

Table 6-35. 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 6-36. Variables to select preferred Lua versions

Name	Designed for
WANT_LUA_VER	the port
WITH_LUA_VER	the user

Example 6-8. 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 using `WITH_LUA_VER`.

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

6.18.3 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 6-37. Available Lua components

Name	Description	Version restriction
lua	main library	none
tolua	Library for accesing 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 6-39). The following types are available:

Table 6-38. 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 6-39. 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 6-9. 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
```

6.18.4 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 6-10. 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) || ${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) || ${HAVE_LUA:Mtolua} != ""
LUA_COMPS+=      tolua
CONFIGURE_ARGS+=--enable-tolua
.endif
```

6.18.5 Defined variables

The following variables are available in the port (after defining one from Table 6-33).

Table 6-40. 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
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 6-11. 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}"
```

6.18.6 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 6-12. 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`.

6.19 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
```

6.20 Using databases

Table 6-41. Variables for ports using databases

Variable	Means
<code>USE_BDB</code>	If variable is set to <code>yes</code> , add dependency on <code>databases/db41</code> port. The variable may also be set to values: 2, 3, 40, 41, 42, 43, 44, 45, 46, or 47. You can declare a range of acceptable values, <code>USE_BDB=42+</code> will find the highest installed version, and fall back to 42 if nothing else is installed.
<code>USE_MYSQL</code>	If variable is set to <code>yes</code> , add dependency on <code>databases/mysql50-server</code> port. An associated variable, <code>WANT_MYSQL_VER</code> , may be set to values such as 323, 40, 41, 50, 51 or 60.
<code>USE_PGSQL</code>	If set to <code>yes</code> , add dependency on <code>databases/postgresql82</code> port. An associated variable, <code>WANT_PGSQL_VER</code> , may be set to values such as 73, 74, 80, 81, 82, or 83.

6.21 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

(http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/configtuning-rcd.html). Detailed explanation of available commands is provided in `rc(8)` and `rc.subr(8)`. Finally, there is an article (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/rc-scripting) on practical aspects of `rc.d` scripting.

One or more `rc` 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. The only difference from a base system `rc.d` script is that the `. /etc/rc.subr` line must be replaced with the `. %%RC_SUBR%%`, because older versions of FreeBSD do not have an `/etc/rc.subr` file. Standard `SUB_LIST`

expansions are used too. 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 deprecated.

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

# PROVIDE: doormand
# REQUIRE: LOGIN
#
# 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.
#

. %%RC_SUBR%%

name="doormand"
rcvar=${name}_enable

command=%%PREFIX%%/sbin/${name}
pidfile=/var/run/${name}.pid

load_rc_config $name

: ${doormand_enable="NO"}
: ${doormand_config="%%PREFIX%%/etc/doormand/doormand.cf"}

command_args="-p $pidfile -f $doormand_config"

run_rc_command "$1"
```

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 suffix of the `rc` script is provided in `RC_SUBR_SUFFIX` for further use in the port's `Makefile`. Current versions of FreeBSD do not add any suffix to the script name, but older versions used to add `.sh` suffix.

Note: No new scripts should be added with the `.sh` suffix. At some point there will be a mass repocopy of all the scripts that still have that suffix.

6.21.1 Stopping services at deinstall

It is possible to have a service stopped automatically as part of the deinstall routine. We advise using this feature only when it's absolutely necessary to stop a service before its files go away. Usually, it's up to the administrator's discretion to decide, whether to stop the service on deinstall or not. Also note this affects upgrades, too.

Line like this goes to the `pkg-plist`:

```
@stopdaemon doormand
```

The argument must match the content of `USE_RC_SUBR` variable.

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 `%%VAR%%` 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`.

This substitution (as well as addition of any manual pages) will be done between the `pre-install` and `do-install` targets, by reading from `PLIST` and writing to `TMPLIST` (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 `TMPLIST`.

Another possibility to modify port's packing list is based on setting the variables `PLIST_FILES` and `PLIST_DIRS`. The value of each variable is regarded as a list of pathnames to write to `TMPLIST` along with `PLIST` contents. Names listed in `PLIST_FILES` and `PLIST_DIRS` 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` will be prepended to names from `PLIST_DIRS`. To take effect, `PLIST_FILES` and `PLIST_DIRS` must be set before `TMPLIST` 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 requires some configuration files in `PREFIX/etc`, do *not* just install them and list them in `pkg-plist`. That will cause `pkg_delete(1)` to delete files carefully edited by the user and a new installation to wipe them out.

Instead, install sample files with a suffix (*filename.sample* will work well). Copy the sample file as the real configuration file, if it does not exist. On deinstall, delete the configuration file, but only if it was not modified by the user. 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 ${PREFIX}/etc/orbit.conf ; \
fi
```

Example of the `pkg-plist` part:

```
@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/etc/orbit.conf; fi
```

Alternatively, print out a message pointing out that the user has to copy and edit the file before the software can be made to work.

7.4 Dynamic vs. 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` and `PLIST_DIRS`. 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**).

Maintainers who prefer dynamic package lists are encouraged to add a new target to their port which generates the `pkg-plist` file so that users may examine the contents.

7.5 Automated package list creation

First, make sure your port is almost complete, with only `pkg-plist` missing.

Next, create a temporary directory tree into which your port can be installed, and install any dependencies.

```
# mkdir /var/tmp/$(make -V PORTNAME)
# mtree -U -f $(make -V MTREE_FILE) -d -e -p /var/tmp/$(make -V PORTNAME)
# make depends PREFIX=/var/tmp/$(make -V PORTNAME)
```

Store the directory structure in a new file.

```
# (cd /var/tmp/$(make -V PORTNAME) && find -d * -type d) | sort > OLD-DIRS
```

Create an empty `pkg-plist` file:

```
# :>pkg-plist
```

If your port honors `PREFIX` (which it should) you can then install the port and create the package list.

```
# make install PREFIX=/var/tmp/$(make -V PORTNAME)
# (cd /var/tmp/$(make -V PORTNAME) && find -d * \! -type d) | sort > pkg-plist
```

You must also add any newly created directories to the packing list.

```
# (cd /var/tmp/$(make -V PORTNAME) && find -d * -type d) | sort | comm -13 OLD-DIRS - | sort -r | sed -e 's/^#@
```

Finally, you need to tidy up the packing list by hand; it is not *all* automated. Manual pages should be listed in the port's `Makefile` under `MANn`, and not in the package list. 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.

Alternatively, use the `plist` script in `/usr/ports/Tools/scripts/` to build the package list automatically. The `plist` script is a **Ruby** script that automates most of the manual steps outlined in the previous paragraphs.

The first step is the same as above: take the first three lines, that is, `mkdir`, `mtree` and `make depends`. Then build and install the port:

```
# make install PREFIX=/var/tmp/$(make -V PORTNAME)
```

And let `plist` create the `pkg-plist` file:

```
# /usr/ports/Tools/scripts/plist -Md -m $(make -V MTREE_FILE) /var/tmp/$(make -V PORTNAME) > pkg-plist
```

The packing list still has to be tidied up by hand as stated above.

Another tool that might be used to create an initial `pkg-plist` is `ports-mgmt/genplist`. As with any automated tool, the resulting `pkg-plist` should be checked and manually edited as needed.

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.

A good example for both can be found in `shells/bash2/Makefile`:

```
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`. Also, it will not get automatically printed if the user is using the port, not the package, so you should probably display it from the `post-install` target yourself.

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 `pkg-req`

If your port needs to determine if it should install or not, you can create a `pkg-req` “requirements” script. It will be invoked automatically at installation/de-installation time to determine whether or not installation/de-installation should proceed.

The script will be run at installation time by `pkg_add(1)` as `pkg-req ${PKGNAME} INSTALL`. At de-installation time it will be run by `pkg_delete(1)` as `pkg-req ${PKGNAME} DEINSTALL`.

8.5 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 `${MASTERDIR}`.)

Variable	Default value
<code>DESCR</code>	<code>\${PKGDIR}/pkg-descr</code>
<code>PLIST</code>	<code>\${PKGDIR}/pkg-plist</code>
<code>PKGINSTALL</code>	<code>\${PKGDIR}/pkg-install</code>
<code>PKGDEINSTALL</code>	<code>\${PKGDIR}/pkg-deinstall</code>
<code>PKGREQ</code>	<code>\${PKGDIR}/pkg-req</code>
<code>PKGMESSAGE</code>	<code>\${PKGDIR}/pkg-message</code>

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.

8.6 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`, `pkg-deinstall` and `pkg-req`, 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`. 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 12.17.)

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 (<http://lists.FreeBSD.org/mailman/listinfo/freebsd-ports>).

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 the location where the port will install. It is usually `/usr/local`, or `/opt`. User can set `PREFIX` to anything he wants. Your port must respect this variable.

`DESTDIR`, if set by user, determines the complete alternative environment, usually a jail, or an installed system mounted elsewhere 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 via calling `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` (default `/usr/local`). If `USE_LINUX_PREFIX` is set, `PREFIX` will be `LINUXBASE` (default `/compat/linux`).

Avoiding the hard-coding of `/usr/local` or `/usr/X11R6` anywhere in the source will make the port much more flexible and able to cater to the needs of other sites. For `X` ports that use `imake`, this is automatic; otherwise, this can often be done by simply replacing the occurrences of `/usr/local` (or `/usr/X11R6` for `X` ports that do not use `imake`) in the various `Makefiles` in the port to read `${PREFIX}`, as 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 this is to do this 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.

This does not test for the existence of internal references, or correct use of `LOCALBASE` for references to files from other ports. Testing the installation in `/var/tmp/${make -V PORTNAME}` to do that while you have it installed would do that.

The variable `PREFIX` can be reassigned in your `Makefile` or in the user's environment. However, it is strongly discouraged for individual ports to set this variable explicitly in the `Makefiles`.

Also, refer to programs/files from other ports with the variables mentioned above, not explicit pathnames. For instance, if your port requires a macro `PAGER` to be the full pathname of `less`, use the compiler flag:

```
-DPAGER=\"${LOCALBASE}/bin/less\"
```

instead of `-DPAGER=\"/usr/local/bin/less\"`. This way it will have a better chance of working if the system administrator has moved the whole `/usr/local` tree somewhere else.

9.5 Tinderbox

If you're an avid ports contributor, you might want to take a look at **Tinderbox**. It is a powerful system for building and testing ports based on the scripts used on Pointyhat. 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 (<http://tinderbox.marcuscom.com/>) for more details.

Chapter 10 Upgrading

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 **CVSup** to keep your whole ports collection up-to-date, as described in the Handbook (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/synching.html#CVSUP-CONFIG). 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 (<http://www.FreeBSD.org/cgi/query-pr-summary.cgi?query>) (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 (<http://portsmon.FreeBSD.org/portoverview.py>).

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 make the changes and save the result of the recursive `diff` 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. 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 2000 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 (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/developers-handbook/policies.html#POLICIES-MAINTAINER).)

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 `cvs(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 (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/problem-reports/pr-writing.html) 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!

Now that you have done all that, you will want to read about how to keep up-to-date in Chapter 14.

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 as a security vulnerability is discovered 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 just 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 by the time it comes to really serious matters. Therefore security vulnerabilities found in ports are recorded in the FreeBSD VuXML database (<http://vuxml.freebsd.org/>). The Security Officer Team members are monitoring 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, or whatever, please do not hesitate to contact the Security Officer Team directly as described on the FreeBSD Security Information (<http://www.freebsd.org/security/#how>) page.

All right, you elected the hard way. As it may be obvious from its title, the VuXML database is essentially 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/security/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 as a template.

11.3.2 A short introduction to VuXML

The full-blown XML 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. Sounds very similar to HTML, doesn't it? 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 let's 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> ❶
  <range><lt>1.10_7</lt></range> ❷
  <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> ❸
    <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> ❹
  <freebsdsv>SA-10:75.foo</freebsdsv> ❺
  <freebsdpr>ports/987654</freebsdpr> (10)
  <cvename>CAN-2010-0201</cvename> (11)
  <cvename>CAN-2010-0466</cvename>
  <bid>96298</bid> (12)
  <certsa>CA-2010-99</certsa> (13)
  <certvu>740169</certvu> (14)
  <uscertsa>SA10-99A</uscertsa> (15)
  <uscertta>SA10-99A</uscertta> (16)
  <mlist msgid="201075606@hacker.com">http://marc.theaimsgroup.com/?l=bugtraq&m=2038866078256
  <url>http://j.r.hacker.com/advisories/1</url> (18)
</references>
<dates>
  <discovery>2010-05-25</discovery> (19)
  <entry>2010-07-13</entry> (20)
  <modified>2010-09-17</modified> (21)
</dates>
</vuln>

```

The tag names are supposed to be self-descriptive, 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; alternatively, if you are using FreeBSD 4.x, you may install the port `devel/p5-Data-UUID` and issue the following command:


```
perl -MData::UUID -le 'print lc new Data::UUID->create_str'
```
- 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.

- ④ 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 obviously 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`.

- ⑤ 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.
- ⑥ 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`.
- ⑦ 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.
- ⑧ This section contains references to relevant documents. As many references as apply are encouraged.
- ⑨ This is a FreeBSD security advisory (<http://www.freebsd.org/security/#adv>).
- (10) This is a FreeBSD problem report (<http://www.freebsd.org/support.html#gnats>).
- (11) This is a Mitre CVE (<http://www.cve.mitre.org/>) identifier.
- (12) This is a SecurityFocus Bug ID (<http://www.securityfocus.com/bid>).
- (13) This is a US-CERT (<http://www.cert.org/>) security advisory.
- (14) This is a US-CERT (<http://www.cert.org/>) vulnerability note.
- (15) This is a US-CERT (<http://www.cert.org/>) Cyber Security Alert.
- (16) This is a US-CERT (<http://www.cert.org/>) Technical Cyber Security Alert.
- (17) 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.
- (18) This is a generic URL. It should be used only if none of the other reference categories apply.
- (19) This is the date when the issue was disclosed (`YYYY-MM-DD`).
- (20) 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` and `ports-mgmt/portaudit-db`.

First, check whether there already is an entry for this vulnerability. If there were such entry, it would match the previous version of the package, `0.65_6`:

```
% packaudit
% portaudit clamav-0.65_6
```

Note: To run `packaudit`, you must have permission to write to its `DATABASEDIR`, typically `/var/db/portaudit`.

If there is none found, you get the green light to add a new entry for this vulnerability. Now you can generate a brand-new UUID (assume it's `74a9541d-5d6c-11d8-80e3-0020ed76ef5a`) and add your new entry to the VuXML database. Please verify its syntax after that as follows:

```
% cd ${PORTSDIR}/security/vuxml && 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 74a9541d-5d6c-11d8-80e3-0020ed76ef5a
```

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.
```


Obviously, 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 (<http://www.FreeBSD.org/cgi/query-pr-summary.cgi?query>) that others have submitted. Submit any comments on ports you check as described in Bug Reports and General Commentary (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/contributing/contrib-how.html#CONTRIB-GENERAL). 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 (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/ports-using.html#PORTS-CD) 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 Unix it is running under. If you need to make such changes to the code for conditional compilation, make sure you make the changes as general as possible so that we can back-port code to older FreeBSD systems and cross-port to other BSD systems such as 4.4BSD from CSRG, BSD/386, 386BSD, NetBSD, and OpenBSD.

The preferred way to tell 4.3BSD/Reno (1990) and newer versions of the BSD code apart is by using the `BSD` macro defined in `sys/param.h` (<http://cvsweb.freebsd.org/src/sys/sys/param.h>). Hopefully that file is already included; if not, add the code:

```
#if (defined(__unix__) || defined(unix)) && !defined(USG)
#include <sys/param.h>
#endif
```

to the proper place in the `.c` file. We believe that every system that defines these two symbols has `sys/param.h`. If you find a system that does not, we would like to know. Please send mail to the FreeBSD ports mailing list (<http://lists.FreeBSD.org/mailman/listinfo/freebsd-ports>).

Another way is to use the GNU Autoconf style of doing this:

```
#ifdef HAVE_SYS_PARAM_H
#include <sys/param.h>
#endif
```

Do not forget to add `-DHAVE_SYS_PARAM_H` to the `CFLAGS` in the Makefile for this method.

Once you have `sys/param.h` included, you may use:

```
#if (defined(BSD) && (BSD >= 199103))
```

to detect if the code is being compiled on a 4.3 Net2 code base or newer (e.g. FreeBSD 1.x, 4.3/Reno, NetBSD 0.9, 386BSD, BSD/386 1.1 and below).

Use:

```
#if (defined(BSD) && (BSD >= 199306))
```

to detect if the code is being compiled on a 4.4 code base or newer (e.g. FreeBSD 2.x, 4.4, NetBSD 1.0, BSD/386 2.0 or above).

The value of the `BSD` macro is 199506 for the 4.4BSD-Lite2 code base. This is stated for informational purposes only. It should not be used to distinguish between versions of FreeBSD based only on 4.4-Lite vs. versions that have merged in changes from 4.4-Lite2. The `__FreeBSD__` macro should be used instead.

Use sparingly:

- `__FreeBSD__` is defined in all versions of FreeBSD. Use it if the change you are making *only* affects FreeBSD. Porting gotchas like the use of `sys_errlist[]` vs `strerror()` are Berkeley-isms, not FreeBSD changes.
- In FreeBSD 2.x, `__FreeBSD__` is defined to be 2. In earlier versions, it is 1. Later versions always bump it to match their major version number.
- If you need to tell the difference between a FreeBSD 1.x system and a FreeBSD 2.x or above system, usually the right answer is to use the `BSD` macros described above. If there actually is a FreeBSD specific change (such as special shared library options when using `ld`) then it is OK to use `__FreeBSD__` and `#if __FreeBSD__ > 1` to detect a FreeBSD 2.x and later system. If you need more granularity in detecting FreeBSD systems since 2.0-RELEASE you can use the following:

```
#if __FreeBSD__ >= 2
#include <osreldate.h>
#   if __FreeBSD_version >= 199504
/* 2.0.5+ release specific code here */
#   endif
#endif
```

In the hundreds of ports that have been done, there have only been one or two cases where `__FreeBSD__` should have been used. Just because an earlier port screwed up and used it in the wrong place does not mean you should do so too.

12.5 __FreeBSD_version values

Here is a convenient list of __FreeBSD_version values as defined in sys/param.h (<http://cvswweb.freebsd.org/src/sys/sys/param.h>):

Table 12-1. __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
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

Value	Date	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
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
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

Value	Date	Release
400010	September 25, 1999	4.0-CURRENT after fixing a hole in jail(2)
400011	September 29, 1999	4.0-CURRENT after the <code>sigset_t</code> 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 <code>-fvtable-thunks</code> to <code>-fno-vtable-thunks</code> 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 branding 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 <code>wint_t</code> 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 <code>d_thread_t</code> introduction.

Value	Date	Release
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 <code>sendmail_enable rc.conf(5)</code> variable was made to take the value <code>NONE</code> .
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 <code>sendfile(2)</code> 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 <code>pkg_install</code> features from the HEAD.
470000	October 8, 2002	4.7-RELEASE
470100	October 9, 2002	4.7-STABLE
470101	November 10, 2002	Start generated <code>__std{in,out,err}p</code> references rather than <code>__sF</code> . This changes <code>std{in,out,err}</code> from a compile time expression to a runtime one.
470102	January 23, 2003	4.7-STABLE after MFC of <code>mbuf</code> changes to replace <code>m_aux</code> mbufs by <code>m_tag</code> 's
470103	February 14, 2003	4.7-STABLE gets OpenSSL 0.9.7
480000	March 30, 2003	4.8-RELEASE
480100	April 5, 2003	4.8-STABLE
480101	May 22, 2003	4.8-STABLE after <code>realpath(3)</code> has been made thread-safe
480102	August 10, 2003	4.8-STABLE 3ware API changes to <code>twe</code> .
490000	October 27, 2003	4.9-RELEASE

Value	Date	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.
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.

Value	Date	Release
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 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.
(not changed)		5.0-CURRENT after the new C99 specific-width integer types were added.

Value	Date	Release
500029	January 29, 2002	5.0-CURRENT after a change was made in the return value of <code>sendfile(2)</code> .
500030	February 15, 2002	5.0-CURRENT after the introduction of the type <code>fflags_t</code> , which is the appropriate size for file flags.
500031	February 24, 2002	5.0-CURRENT after the <code>usb</code> 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 <code>sendmail_enable rc.conf(5)</code> variable was made to take the value <code>NONE</code> .
500034	April 30, 2002	5.0-CURRENT after <code>mtx_init()</code> 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 <code>/usr/src</code>
500037	May 29, 2002	5.0-CURRENT after the addition of <code>dlfunc(3)</code>
500038	July 24, 2002	5.0-CURRENT after the types of some struct <code>sockbuf</code> 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 <code>_BSD_FOO_T_</code> and started using <code>_FOO_T_DECLARED</code> . This value can also be used as a conservative estimate of the start of <code>bzip2(1)</code> package support.
500040	September 20, 2002	5.0-CURRENT after various changes to disk functions were made in the name of removing dependency on <code>disklabel</code> structure internals.
500041	October 1, 2002	5.0-CURRENT after the addition of <code>getopt_long(3)</code> to <code>libc</code> .
500042	October 15, 2002	5.0-CURRENT after Binutils 2.13 upgrade, which included new FreeBSD emulation, <code>vec</code> , and output format.
500043	November 1, 2002	5.0-CURRENT after adding weak <code>pthread_XXX</code> stubs to <code>libc</code> , obsoleting <code>libXThrStub.so</code> . 5.0-RELEASE.

Value	Date	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.
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.

Value	Date	Release
501104	August 5, 2003	5.1-CURRENT 3ware API changes to two.
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.
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.

Value	Date	Release
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.
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.

Value	Date	Release
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 <code>xorg</code> was also made at this time.
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 backout of the previous change.
502126	July 31, 2004	5.2-CURRENT after the removal of <code>kmem_alloc_pageable()</code> 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 <code>vfs_mount</code> signature as well as global replacement of PRISON_ROOT with SUSER_ALLOWJAIL for the <code>suser(9)</code> 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 <code>strftime(3)</code> padding options.
503102	February 13, 2005	5.3-STABLE after OpenBSD's <code>nc(1)</code> import MFC.
503103	February 27, 2005	5.4-PRERELEASE after the MFC of the fixes in <code><src/include/stdbool.h></code> and <code><src/sys/i386/include/_types.h></code> for using the GCC-compatibility of the Intel C/C++ compiler.

Value	Date	Release
503104	February 28, 2005	5.4-PRERELEASE after the MFC of the change of 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 stack sizes
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.
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.

Value	Date	Release
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 <code><src/include/stdbool.h></code> and <code><src/sys/i386/include/_types.h></code> for using the GCC-compatibility of the Intel C/C++ compiler.
600014	February 21, 2005	6.0-CURRENT after EOVERFLOW checks in vsprintf(3) fixed.
600015	February 25, 2005	6.0-CURRENT after changing the struct if_data member, ifi_epoch, from wall clock time to uptime.
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.

Value	Date	Release
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.
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) support has been enabled in OpenSSL.
601105	September 2, 2006	6.1-STABLE after 802.11 fixups changed the api for the IEEE80211_IOC_STA_INFO ioctl.
602000	November 15, 2006	6.2-RELEASE

Value	Date	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.
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.

Value	Date	Release
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.
603103	May 27, 2008	6.3-STABLE after MFC of the <code>memrchr</code> function.
603104	June 15, 2008	6.3-STABLE after MFC of support for <code>:u</code> variable modifier in <code>make(1)</code> .
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 <code>RELENG_5</code> .
700002	August 13, 2005	7.0-CURRENT after credential argument is added to <code>dev_clone</code> event handler.
700003	August 25, 2005	7.0-CURRENT after <code>memmem(3)</code> is added to <code>libc</code> .
700004	October 30, 2005	7.0-CURRENT after <code>solisten(9)</code> kernel arguments are modified to accept a backlog parameter.
700005	November 11, 2005	7.0-CURRENT after <code>IFP2ENADDR()</code> was changed to return a pointer to <code>IF_LLADDR()</code> .
700006	November 11, 2005	7.0-CURRENT after addition of <code>if_addr</code> member to <code>struct ifnet</code> and <code>IFP2ENADDR()</code> removal.
700007	December 2, 2005	7.0-CURRENT after incorporating scripts from the <code>local_startup</code> directories into the <code>base rcorder(8)</code> .
700008	December 5, 2005	7.0-CURRENT after removal of <code>MNT_NODEV</code> mount option.
700009	December 19, 2005	7.0-CURRENT after ELF-64 type changes and symbol versioning.

Value	Date	Release
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.
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.

Value	Date	Release
700028	December 15, 2006	7.0-CURRENT after the addition of setp 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) firmwares.
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(), getnewvnode(), 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.

Value	Date	Release
700044	June 7, 2007	7.0-CURRENT after changing the argument for <code>vn_open()/VOP_OPEN()</code> from filedescriptor index to the struct file *.
700045	June 10, 2007	7.0-CURRENT after changing <code>pam_nologin(8)</code> to provide an account management 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 <code>IP_MULTICAST_IF</code> ioctl has now been removed; 0.0.0.0/8 may no longer be used to specify an interface index. struct <code>ipmreqn</code> 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 <code>setenv/putenv/etc.</code> calls from traditional BSD to POSIX.
700051	July 4, 2007	7.0-CURRENT after adding new <code>mmap/lseek/etc</code> 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
700054	October 25, 2007	7.0-CURRENT after MFC of wide and single byte ctype separation.

Value	Date	Release
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 MFC'ed, 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 <code>m_collapse()</code> .
700102	March 30, 2008	7.0-STABLE after the MFC of <code>kdb_enter_why()</code> .
700103	April 10, 2008	7.0-STABLE after adding <code>l_sysid</code> to <code>struct flock</code> .
700104	April 11, 2008	7.0-STABLE after the MFC of <code>procstat(1)</code> .
700105	April 11, 2008	7.0-STABLE after the MFC of <code>umtx</code> features.
700106	April 15, 2008	7.0-STABLE after the MFC of <code>write(2)</code> support to <code>psm(4)</code> .
700107	April 20, 2008	7.0-STABLE after the MFC of <code>F_DUP2FD</code> command to <code>fcntl(2)</code> .
700108	May 5, 2008	7.0-STABLE after some <code>lockmgr(9)</code> changes, which makes it necessary to include <code>sys/lock.h</code> in order to use <code>lockmgr(9)</code> .
700109	May 27, 2008	7.0-STABLE after MFC of the <code>memrchr</code> function.
700110	August 5, 2008	7.0-STABLE after MFC of kernel NFS <code>lockd</code> 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 <code>strndup</code> merge.
701102	January 17, 2009	7.1-STABLE after <code>cpuctl(4)</code> support added.
701103	February 7, 2009	7.1-STABLE after the merge of multi-/no-IPv4/v6 jails.

Value	Date	Release
701104	February 14, 2009	7.1-STABLE after the store of the suspension owner in the struct mount, and introduction of <code>vfs_susp_clean</code> method into the struct <code>vfsops</code> .
701105	March 12, 2009	7.1-STABLE after the incompatible change to the <code>kern.ipc.shmseg</code> 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 <code>ichsmb(4)</code> was changed to use left-adjusted slave addressing to match other SMBus controller drivers.
702102	May 28, 2009	7.2-STABLE after MFC of the <code>fdopendir</code> function.
702103	June 06, 2009	7.2-STABLE after MFC of <code>PmcTools</code> .
702104	July 14, 2009	7.2-STABLE after MFC of the <code>closefrom</code> system call.
702105	July 31, 2009	7.2-STABLE after MFC of the SYSVIPIC ABI change.
800000	October 11, 2007	8.0-CURRENT. Separating wide and single byte ctype.
800001	October 16, 2007	8.0-CURRENT after <code>libpcap 0.9.8</code> and <code>tcpdump 3.9.8</code> import.
800002	October 21, 2007	8.0-CURRENT after renaming <code>kthread_create()</code> and friends to <code>kproc_create()</code> etc.
800003	October 24, 2007	8.0-CURRENT after ABI backwards compatibility to the FreeBSD 4/5/6 versions of the <code>PCIOCGETCONF</code> , <code>PCIOCREAD</code> and <code>PCIOCWRITE</code> IOCTLs was added, which required the ABI of the <code>PCIOCGETCONF</code> IOCTL to be broken again
800004	November 12, 2007	8.0-CURRENT after <code>agp(4)</code> driver moved from <code>src/sys/pci</code> to <code>src/sys/dev/agp</code>

Value	Date	Release
800005	December 4, 2007	8.0-CURRENT after changes to the jumbo frame allocator (http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/kern/kern
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_recurse(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) 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.

Value	Date	Release
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 return 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.

Value	Date	Release
800032	April 8, 2008	8.0-CURRENT after the implementation of the <code>openat</code> and related syscalls, introduction of the <code>O_EXEC</code> flag for the <code>open(2)</code> , and providing the corresponding linux compatibility syscalls.
800033	April 8, 2008	8.0-CURRENT after added <code>write(2)</code> support for <code>psm(4)</code> in native operation level. Now arbitrary commands can be written to <code>/dev/psm%d</code> and status can be read back from it.
800034	April 10, 2008	8.0-CURRENT after introduction of the <code>memrchr</code> function.
800035	April 16, 2008	8.0-CURRENT after introduction of the <code>fdopendir</code> 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 (a.k.a. <code>setfib(1)</code> , <code>setfib(2)</code>).
800038	May 26, 2008	8.0-CURRENT after removal of <code>netatm</code> and <code>ISDN4BSD</code> .
800039	June 14, 2008	8.0-CURRENT after removal of <code>sgtty</code> .
800040	June 26, 2008	8.0-CURRENT with kernel NFS <code>lockd</code> client.
800041	July 22, 2008	8.0-CURRENT after addition of <code>arc4random_buf(3)</code> and <code>arc4random_uniform(3)</code> .
800042	August 8, 2008	8.0-CURRENT after addition of <code>cpuctl(4)</code> .
800043	August 13, 2008	8.0-CURRENT after changing <code>bpf(4)</code> 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 <code>vimage</code> project renaming global variables to be virtualized with a <code>V_</code> 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.

Value	Date	Release
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.
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.

Value	Date	Release
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.
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(), strlen(), wcsnlen(), wscasecmp(), 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 pathes.
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 and vnet_ipfw has been changed.

Value	Date	Release
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.
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.

Value	Date	Release
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 <code>sx_init_flags()</code> KPI. The <code>SX_ADAPTIVESPIN</code> is retired and a new <code>SX_NOADAPTIVE</code> flag is introduced in order to handle the reversed logic.
800093	May 29, 2009	8.0-CURRENT after adding <code>mnt_xflag</code> to struct mount.
800094	May 30, 2009	8.0-CURRENT after adding <code>VOP_ACCESSX(9)</code> .
800095	May 30, 2009	8.0-CURRENT after changing the polling KPI. The polling handlers now return the number of packets processed. A new <code>IFCAP_POLLING_NOCOUNT</code> is also introduced to specify 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 <code>/dev/net/*</code> per-interface character devices.

Value	Date	Release
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 tcptopt with struct toept in the TOE driver interface to the TCP syncache.
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.
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.

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.

12.6 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: You need to 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 <code>uname -m</code> (e.g., <code>i386</code>)
OPSYS	The operating system type, as returned by <code>uname -s</code> (e.g., <code>FreeBSD</code>)
OSREL	The release version of the operating system (e.g., <code>2.1.5</code> or <code>2.2.7</code>)
OSVERSION	The numeric version of the operating system; the same as <code>__FreeBSD_version</code> .
PORTOBJFORMAT	The object format of the system (<code>elf</code> or <code>aout</code> ; note that for “modern” versions of <code>FreeBSD</code> , <code>aout</code> is deprecated.)
LOCALBASE	The base of the “local” tree (e.g., <code>/usr/local/</code>)
PREFIX	Where the port installs itself (see more on <code>PREFIX</code>).

Note: If you have to define the variables `USE_IMAKE`, `USE_X_PREFIX`, or `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

# only one shlib version number for ELF
.if ${PORTOBJFORMAT} == "elf"
TCL_LIB_FILE= ${TCL_LIB}.${SHLIB_MAJOR}
.else
TCL_LIB_FILE= ${TCL_LIB}.${SHLIB_MAJOR}.${SHLIB_MINOR}
.endif

# software already makes link for ELF, but not for a.out
post-install:
.if ${PORTOBJFORMAT} == "aout"
${LN} -sf liblinpack.so.1.0 ${PREFIX}/lib/liblinpack.so
.endif
```

You did remember to use `tab` instead of `spaces` after `BROKEN=` and `TCL_LIB_FILE=`, did you not? :-).

12.7 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.

12.8 UIDs and GIDs

The current list of reserved UIDs and GIDs can be found in `ports/UIDs` and `ports/GIDs`.

If your port requires a certain user to be on the installed system, let the `pkg-install` script call `pw` to create it automatically. Look at `sysutils/symon` for an example. Your port must use a fixed user/group ID number. You must 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.

12.9 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.

12.10 Respect both `CC` and `CXX`

The port should respect both `CC` and `CXX` variables. What we mean by this is that the port should not set the values of these variables absolutely, overriding existing values; instead, it should 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.

12.11 Respect `CFLAGS`

The port should respect the `CFLAGS` variable. What we mean by this is that the port should not set the value of this variable absolutely, overriding the existing value; instead, it should 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
```

12.12 Threading libraries

The threading library must be linked to the binaries using a special linker flag `-pthread` on FreeBSD. If a port insists on linking `-lpthread` or `-lc_r` directly, patch it to use `PTHREAD_LIBS` variable provided by the ports framework. This variable usually has the value of `-pthread`, but on certain architectures and FreeBSD versions it can have different values, so do not just hardcode `-pthread` into patches and always use `PTHREAD_LIBS`.

Note: If building the port errors out with unrecognized option `'-pthread'` when setting `PTHREAD_LIBS`, it may be desirable to use `gcc` as linker by setting `CONFIGURE_ENV` to `LD=${CC}`. The `-pthread` option is not supported by `ld` directly.

12.13 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.

12.14 README.html

Do not include the `README.html` file. This file is not part of the cvs collection but is generated using the `make readme` command.

12.15 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.

12.15.1 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 do contain a security vulnerability or induce grave concern regarding the security of a FreeBSD system with a given port installed (ex: a reputedly 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
 - requires FreeBSD kernel sources to build, but the user does not have them installed
 - 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/apache21` but `www/apache13` 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= alpha 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.

12.15.2 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= this port is unsupported on FreeBSD 5.x
```

```
IGNORE= is unsupported on FreeBSD 5.x
```

resulting in the following output from `make describe`:

```
==> foobar-0.1 is marked as broken: this port is unsupported on FreeBSD 5.x.
```

```
==> foobar-0.1 is unsupported on FreeBSD 5.x.
```

12.16 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.

12.17 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 nonblank 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). However, even more trivial commands such as `make -V maintainer` also fail in this scenario. This is not acceptable.

Example 12-1. How to avoid using `.error`

Assume that someone has the line

```
USE_POINTYHAT=yes
```

in `make.conf`. The first of the next two `Makefile` snippets will cause `make index` to fail, while the second one will not:

```
.if USE_POINTYHAT
.error "POINTYHAT is not supported"
	endif
.if USE_POINTYHAT
IGNORE=POINTYHAT is not supported
	endif
```

12.18 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.

12.19 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.

12.20 Necessary workarounds

Sometimes it is necessary to work around bugs in software included with older versions of FreeBSD.

- Some versions of `make(1)` were broken on at least 4.8 and 5.0 with respect to handling comparisons based on `OSVERSION`. This would often lead to failures during `make describe` (and thus, the overall ports `make index`). The workaround is to enclose the conditional comparison in spaces, e.g.:

```
if ( ${OSVERSION} > 500023 )
```

Be aware that test-installing a port on 4.9 or 5.2 will *not* detect this problem.

12.21 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.]
# New ports collection makefile for:  xdvi
[the "version required" line is only needed when the PORTVERSION
variable is not specific enough to describe the port.]
# Date created:                26 May 1995
[this is the person who did the original port to FreeBSD, in particular, the
person who wrote the first version of this Makefile. Remember, this should
not be changed when upgrading the port later.]
# Whom:                        Satoshi Asami <asami@FreeBSD.org>
#
# $FreeBSD$
[ ^^^^^^^^^ This will be automatically replaced with RCS ID string by CVS
when it is committed to our repository. If upgrading a port, do not alter
this line back to "$FreeBSD$". CVS 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=       A DVI Previewer for the X Window System

[dependencies -- can be empty]
RUN_DEPENDS=   gs:${PORTSDIR}/print/ghostscript
LIB_DEPENDS=   Xpm.5:${PORTSDIR}/graphics/xpm

[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...]
USE_GMAKE=    yes
[If it is an X application and requires "xmkmf -a" to be run...]
USE_IMAKE=    yes
[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 (<http://www.FreshPorts.org/>). 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` (<http://cvsweb.FreeBSD.org/ports/CHANGES>) file. Changes that affect individual ports are now documented in the `UPDATING` (<http://cvsweb.FreeBSD.org/ports/UPDATING>) file. However, the definitive answer to any question is undoubtedly to read the source code of `bsd.port.mk` (<http://cvsweb.FreeBSD.org/ports/Mk/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 (<http://lists.FreeBSD.org/mailman/listinfo/freebsd-ports>). Important changes to the way ports work will be announced there, and then committed to `CHANGES`.

14.4 The FreeBSD Port Building Cluster on `pointyhat.FreeBSD.org`

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. You can find the results of these builds at package building logs and errors (<http://pointyhat.FreeBSD.org/>).

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 The FreeBSD Port Distfile Survey

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. The FreeBSD Ports distfiles survey (<http://people.FreeBSD.org/~fenner/portsurvey/>) attempts to query every download site for every port to find out if each distfile is still currently available. Maintainers are asked to check this report periodically, not only to speed up the building process for users, but to help avoid wasting bandwidth of the sites that volunteer to host all these distfiles.

14.6 The FreeBSD Ports Monitoring System

Another handy resource is the FreeBSD Ports Monitoring System (<http://portsmon.FreeBSD.org>) (also known as `portsmon`). This system comprises a database that processes information from several sources and allows its 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 collection 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 (<http://portsmon.FreeBSD.org/portoverview.py>).

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.