




Prentice Hall Open Source Software Development Series

Using BusyBox

Christopher Hallinan

11.1 Introduction to BusyBox	3
11.1.1 BusyBox is Easy	4
11.2 BusyBox Configuration	5
11.2.1 Cross-Compiling BusyBox	7
11.3 BusyBox Operation	8
11.3.1 BusyBox Init	12
11.3.2 Example rcS Initialization Script ..	16
11.3.3 BusyBox Target Installation	17
11.3.4 BusyBox Commands	20
11.4 Chapter Summary	21
11.4.1 Suggestions for Additional	22
Reading	





Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this work, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author(s) and publisher have taken care in the preparation of this work, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Visit us on the Web: www.prenhallprofessional.com

Copyright © 2007 Pearson Education, Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
One Lake Street
Upper Saddle River, NJ 07458
United States of America
Fax: (201)236-3290

ISBN 0-13-233592-1

First release, July 2006

Notice that the BusyBox utility, as compiled using the default configuration, requires the three shared libraries in Listing 11-2. Had we elected to build BusyBox as a static binary, `ldd` would simply issue a message telling us that the BusyBox binary is not a dynamic executable. In other words, it requires no shared libraries to resolve any unresolved dependencies in the executable. Static linking yields a smaller footprint on a root file system because no shared libraries are required. However, building an embedded application without shared libraries means that you have none of the familiar C library functions available to your applications.

We cover the other options from Listing 11-1 in the next section.

11.2.1 Cross-Compiling BusyBox

As mentioned at the beginning of the chapter, the authors of BusyBox intended the package to be used in a cross-development environment, so building BusyBox in such an environment is quite easy. In most cases, the only requirement is to specify the prefix to the cross-compiler on your development workstation. This is specified in Build Options in the BusyBox configuration utility by selecting the option to build BusyBox with a cross-compiler. You then are presented with an option to enter the cross-compiler prefix. The prefix you enter depends on your cross-development environment. Some examples include `xscale_be-` or `ppc-linux-`. We cover this in more detail in the next chapter when we examine the embedded development environment.

The final option in Listing 11-1 is for any extra flags you might want to include on the compiler command line. These might include options for generating debug information (`-g`), options for setting the optimization level (`-O2`, for example), and other compiler options that might be unique to your particular installation and target system.

11.3 BusyBox Operation

When you build BusyBox, you end up with a binary called, you guessed it, `busybox`. BusyBox can be invoked from the binary name itself, but it is more usually launched via a *symlink*. When BusyBox is invoked without command-line parameters, it produces a list of the functions that were enabled via the configuration. Listing 11-3 shows such an output (it has been formatted slightly to fit the page width).

LISTING 11-3 BusyBox Usage

```
root@coyote # ./busybox
BusyBox v1.01 (2005.12.03-18:00+0000) multi-call binary
```

```
Usage: busybox [function] [arguments]...
       or: [function] [arguments]...
```

```
BusyBox is a multi-call binary that combines many common Unix
utilities into a single executable. Most people will create a
link to busybox for each function they wish to use and BusyBox
will act like whatever it was invoked as!
```

Currently defined functions:

```
[, ash, basename, bunzip2, busybox, bzip, cat, chgrp, chmod,
chown, chroot, chvt, clear, cmp, cp, cut, date, dd, deallocvt,
df, dirname, dmesg, du, echo, egrep, env, expr, false, fgrep,
find, free, grep, gunzip, gzip, halt, head, hexdump, hostname,
```

SECTION 11.3

BusyBox Operation

```
id, ifconfig, init, install, kill, killall, klogd, linuxrc, ln,
logger, ls, mkdir, mknod, mktemp, more, mount, mv, openvt, pidof,
ping, pivot_root, poweroff, ps, pwd, readlink, reboot, reset,
rm, rmdir, route, sed, sh, sleep, sort, strings, swapoff, swapon,
sync, syslogd, tail, tar, tee, test, time, touch, tr, true, tty,
umount, uname, uniq, unzip, uptime, usleep, vi, wc, wget, which,
whoami, xargs, yes, zcat
```

From Listing 11-3, you can see the list of functions that are enabled in this BusyBox build. They are listed in alphabetical order from `ash` (a shell optimized for small memory footprint) to `zcat`, a utility used to decompress the contents of a compressed file. This is the default set of utilities enabled in this particular BusyBox snapshot.

To invoke a particular function, execute `busybox` with one of the defined functions passed on the command line. Thus, to display a listing of files in the current directory, execute this command:

```
[root@coyote]# ./busybox ls
```

Another important message from the BusyBox usage message in Listing 11-3 is the short description of the program. It describes BusyBox as a multical binary, combining many common utilities into a single executable. This is the purpose of the symlinks mentioned earlier. BusyBox was intended to be invoked by a symlink named for the function it will perform. This removes the burden of having to type a two-word command to invoke a given function, and it presents the user with a set of familiar commands for the similarly named utilities. Listings 11-4 and 11-5 should make this clear.