

building software packages  
with sw-utils

Lukas Beeler  
Huber+Monsch

---

November 1, 2003  
*Rev* : 1.2

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Requirements . . . . .	2
1.2	Questions? . . . . .	2
<b>2</b>	<b>Creating your first package</b>	<b>3</b>
2.1	Directory layout . . . . .	3
2.2	Writing the UIB . . . . .	4
2.2.1	Plain UIB . . . . .	4
2.2.2	Example UIB . . . . .	4
2.3	Writing the PIB . . . . .	5
2.3.1	Plain PIB . . . . .	5
2.3.2	Example PIB . . . . .	5
2.4	Writing the PBD . . . . .	6
2.4.1	Plain PBD . . . . .	6
2.4.2	Example PBD . . . . .	6
2.5	Building the package . . . . .	6
<b>3</b>	<b>Converting to other package formats</b>	<b>7</b>
3.1	Debian package format . . . . .	8
3.1.1	Empty control file . . . . .	8
3.1.2	Example control file . . . . .	8
3.1.3	Converting . . . . .	9
<b>A</b>	<b>References</b>	<b>9</b>

## 1 Introduction

This document should explain you how to create a small software package using the sw-utls toolsuite. sw-utls was created with the idea of being able to create packages in a very short amount of time.

### 1.1 Requirements

- general knowledge of bourne shell programming
- general knowledge of compiling software
- general knowledge of your package managment

### 1.2 Questions?

Before asking any questions, make sure you read the manpages of the utilities you intend to use, plus the necessary format specifying manpages (like Pkgfile(5)).

## 2 Creating your first package

We start by choosing the software we want to package. In this example, we have decided to use `bmon`, a small network graphing tool with `ncurses` output, written with the standard `gnu autoconf` toolsuite, which most OSS uses these days.

### 2.1 Directory layout

Usually, we create a directory with the name of the package we intend to build. In our example, the name of this directory is `'bmon'`. This directory contains the following files:

- `bmon/Pkgfile` - package build description file, see `Pkgfile(5)`
- `bmon/.md5sum` - md5sum of sourcefiles
- `bmon/.footprint` - package content footprint, for regress checks

## 2.2 Writing the UIB

The UIB, User Information Block, is the first section of our package build description, called the 'Pkgfile'. This block is meant to be read by a human, not a program. If there are any eventual pitfalls, you are free to mention them in the UIB (or in a separate README shipped with the Pkgfile).

### 2.2.1 Plain UIB

Our default UIB looks like this:

```
# $Id: $
# Description:
# URL:
# Maintainer:
# Depends on:
```

As you can see, most fields are self explanatory, with the exception of the last field. This field mentions the Dependencies of the package build process itself, so you have to look at those dependencies as build dependencies. Remember that UIB is only read by the user, and not by the program (so you won't get a warning about missing dependencies).

### 2.2.2 Example UIB

Now, let's try filling the UIB with the information we need for the bmon package:

```
# $Id: Pkgfile 23 2003-10-29 08:44:23Z lb $
# Description: bandwidth monitor with ncurses and html output
# URL: http://trash.net/~reeler/bmon/
# Maintainer: Lukas Beeler <lukas.beeler@projectdream.org>
# Depends on: glibc, ncurses
```

In our example, \$Id:\$ has already been replaced by the SCMMS we use.

Up to now, it was pretty easy, isn't it?

## 2.3 Writing the PIB

The PIB, Package Information Block, is the second section of our Pkgfile. This part of the Pkgfile is used by most sw-utils, and should thus be handled with far more care. It contains information about name of the package, version, release, and an array containing the urls or filenames needed to build this package.

### 2.3.1 Plain PIB

Our default PIB looks like this:

```
name=  
version=  
release=  
source=( )
```

While name explains itself, version and release require a bit more of attention. version is the version number given by the upstream author of our software, and release is the version number of our package, given by you. The source array (designated as an array by the two brackets) contains all the urls needed to build this package.

### 2.3.2 Example PIB

Let's fill our empty example with some real life values:

```
name=bmon  
version=1.2.0  
release=1  
source=(http://trash.net/~reeler/bmon/files/$name-${version}.tar.bz2)
```

We have reused the already defined variables \$name and \$version while declaring the source array. This is intended to make a version bump much easier.

## 2.4 Writing the PBD

The PBD, Package Build Description, is the last, and perhaps most difficult part of our whole Pkgfile. It contains a shell function that builds and installs our package into the destination root \$PKG. In most cases, this is a pretty easy task, however there are some packages which are somewhat harder to build. Just have a look at the various examples the normal sw collection gives you, or have a look at gentoo's ebuilds.

### 2.4.1 Plain PBD

The default PBD is really boring:

```
build() {
}

```

Nothing interesting, so let's move on to our example.

### 2.4.2 Example PBD

This is the PBD I use for my bmon package:

```
build() {
    cd $name-$version
    ./configure --prefix=/usr --disable-nls

    make

    make DESTDIR=$PKG install
}

```

As you can see, this is pretty easy to do with a little bit of knowledge about shell programming. We reused our \$name and \$version variables again for easier version bumping, and told make to use a DESTDIR of \$PKG. This sample works fine with most GNU autoconf projects. It most probably won't work with other projects, so you will have to read their documentation on how to do this.

## 2.5 Building the package

You can build this package by executing 'fakeroot swmk' in the directory where the Pkgfile resides. Sample output below:

```
=====> Building '/home/lb/src/packages/tarballs/bmon#1.2.0-1.pkg.tar.gz'.
tar [ .. ]
+ build
+ cd bmon-1.2.0
+ ./configure --prefix=/usr --disable-nls
[ .. ]
+ make
[ .. ]
+ make DESTDIR=/home/lb/projects/sw/base/bmon/work/pkg install
=====> Build result:
[ .. ]
=====> Building '/usr/src/packages/tarballs/bmon#1.2.0-1.pkg.tar.gz' succeeded.
```

### 3 Converting to other package formats

We now have built a plain tarball, which you could use by extracting it to /. In most cases, you do not really want to do this, but a package that's in your OS's native format would be nice. `sw-utils` has been created with this in mind. While currently, we only have `swmk_deb` (which produces debian packages), writing something like `swmk.rpm` would be easy.

### 3.1 Debian package format

Debian's package management system isn't the easiest one, but it is very powerfull. Creating a debian package out of a plain tarball is pretty easy. For starters, the only thing you need is a 'control' file, which contains the necessary descriptions and dependency informations.

#### 3.1.1 Empty control file

Look at our empty control file first:

```
Package: @@NAME@@
Version: @@VERSION@@
Section:
Priority:
Installed-Size: @@SIZE@@
Architecture: @@ARCH@@
Depends:
Maintainer:
Description:
```

We already have some values filled in, which are surrounded by two at signs. These are replaced by `swmk_deb` for easier version bumping, architecture porting.

#### 3.1.2 Example control file

This is the control file i use with `bmon`:

```
Package: @@NAME@@
Version: @@VERSION@@
Section: net
Priority: optional
Installed-Size: @@SIZE@@
Architecture: @@ARCH@@
Depends: libncurses5
Maintainer: Lukas Beeler <lb-debian@projectdream.org>
Description: bandwidth monitor
  bmon is an interface bandwidth monitor.
```

```
.
It is able to generate and draw three types of diagrams:
```

- ```
.
* The overview diagram which is a list of all interfaces and their
  send/receive rates.
* A very MRTG like graphical diagram in ASCII showing the rate over a specific
  time period in a bar diagram.
* The details diagram which contains all counters of an interface such as
  total bytes sent/received, errors, compressed packets, ...
```

Still, this doesn't look that difficult. If you need details about the format of the control file (which is pretty self-explanatory), please read the debian documentation.

### 3.1.3 Converting

Converting itself is very easy, just run 'fakeroot swmk\_deb'. For sample output, see below:

```
% fakeroot swmk_deb
=====> Extracting plain tarball
dpkg-deb: building package 'bmon' in 'debian.deb'.
=====> Successfully created /usr/src/packages/debian/bmon_1.2.0-1_i386.deb
```

## A References

Information about sw-utils:

<http://svn.projectdream.org/repos/sw-utils/trunk/man/>

Information about the debian package format:

<http://www.debian.org>