dpkg Programmer's Manual

Klee Dienes

Version 2.1.2.2 (dpkg 1.4.0.12)This manual describes the internal structure of the `dpkg` package management system.

1 Copyright

2 Build Notes

2.1 Automake

This chapter by Tom Lees <tom@lpsg.demon.co.uk> on Tue, 4 Mar 1997 21:34:57 +0000, with subsequent modifications by Klee Dienes <klee@debian.org>

This chapter contains general notes about the conversion to `automake`. If you plan on doing anything with `dpkg`, you should probably read all of this file first. You have been warned.

The current version of automake used is version `automake-1.1n`, which is currently a beta version. It has several significant advantages over the released version, including:

- it supports emacs lisp properly
- It supports libtool properly
- it includes the aclocal utility

The `aclocal` utility is a very useful program which will automatically build an '`aclocal.m4`' file from the '`configure.in`' file to include the appropriate macros.

`automake-1.1n` can be retreived from '`ftp://ftp.cygnus.com/pub/tromey`'

This doesn't affect anything other than rebuilding the '`Makefile.in`' files from the sources.

Probably the main difference which is noticable is that instead of using proprietary directory names, it now supports `configure --sharedstatedir`, and `configure --localstatedir`. To set these to the Debian defaults, you should use `./configure --localstatedir=/etc --sharedstatedir=/var/lib`.

I have also customized the canonicalizing macros found in `autoconf-2.12` to include the old way of finding the `dpkg` "architecture", i.e. to be a bit more smart. Instead of it trying to determine the architecture only, I changed it to use the 'host', 'build', and 'target' system types. The target CPU type is checked against the archtable to find the architecture on which dpkg will run.

It uses `gcc --print-libgcc-file-name` to find out the build architecture if possible (used later to determine ELF or `a.out` format), and also uses `dpkg --print-architecture` if possible to modify the '`cpu`' field before it passes on the target alias to config.sub. If you want to specify the architecture, you should now use "–target=", rather than `--with-arch`, which was essentially a hack anyway. The old `--with-arch` is still there, but it is somewhat less functional. I have also moved the `DPKG_CACHED_` macros into dpkg.m4 to make configure.in a bit more readable.

I also converted to libtool 0.7 (which can be found in the Debian distribution now). Essentially, this means that all the dpkg tools can be compiled against a shared '`libdpkg`' without much hassle (in fact, it is the default). You do not need to install libtool to use this feature (it works like autoconf), and generally, it should not be needed much at all.

The new '`dist`' targets will build a distribution including all files built by the `debiandoc2html`, `debiandoc2ps`, etc., which are included in the distribution so that people may build `dpkg` without these (especially useful to porters).

A target '`make debian`' has been added, which will build the Debian files from a working directory (does a '`make dist`' first). Now all we need is a modified dpkg-source so that the '`dpkg-1.4.0.8.tar.gz`' GNU-distribution file can be used as part of the Debian dsitribution. I'm working on this, but it doesn't work too well at the moment (find it in examples).

I removed the '`make portable`' target - it doesn't do anything useful.

I have added '`make uninstall`' targets to aid non-Debian users who simply want to try out certain Debian packages, and the "dist" targets are also useful to build a "distribution" of the dpkg tool. Note that since automake automatically includes dependencies into the Makefiles in a distribution, if you want to modify the C files, it would be advisable to get and install automake, and then re-run it in the base dpkg distribution directory, so that automatic dependency generation will be switched back on, and any dependencies which change will be taken account of. The "make maintainer-clean" targets will remove all files which any of the following utilities create:

automake

autoconf

aclocal

autoheader

gettextize

libtoolize

If you want to modify any of the sources, I recommend that you do the following first (after having installed the appropriate utilities, of course):-

make maintainer-clean

aclocal

autoheader

autoconf

gettextize

libtoolize (do not let automake run this, as it will cause it not to include the libtool files in the dist targets)

for i in COPYING INSTALL; do ln -s /usr/share/automake/$i .; done

automake

I have also incorporated the patches originally made by Galen Hazelwood to internationalize dpkg using GNU gettext - see the file "NOTES.intl" for more information about this.

Other minor changes are:

The version number is now determined from debian/changelog, not from the directory name.

Creation of version.h now handled from configure script, not Makefile.

include/dpkg.h is now generated from include/dpkg.h.in by a sed script, inserting the appropriate directory definitions - so now it supports changing the dpkg directories (can install in /usr/local)

Updated the COPYING (1 very minor change) and INSTALL files to those distributed with automake-1.1l

Since the shared libdpkg is now installed, I also made include/Makefile install dpkg.h and dpkg-db.h into /usr/include by default

Questions:

Should I use '`localstatedir`' and '`sharedstatedir`' instead of '`sysconfdir`' and '`datadir`'?

2.2 Internationalization

This section by Galen Hazelwood.

Dpkg is, to say the least, generous in its error reporting. The vast majority of the output strings are error messages of one kind or another. And if you feel that you've stumbled into the Department of Redundancy Department, you would be absolutely correct. Many of the error messages in dpkg.pot are duplicates, used at different points in the program.

To avoid swamping the translators completely, I made some executive decisions on what kinds of strings to translate. All the strings sent to debug() are left alone, on the grounds that these are for dpkg developers, and not for the general public. Most interal error messages were very cryptic, and would probably confuse the translators when seen just sitting there in the dpkg.pot file, and are also left alone. (I did mark some of the more verbose ones for translation.)

If others disagree with me about the necessity of translating these strings, it's easy enough to just go through and mark them later.

I added the startup gettext code to the main routine in dselect, which was necessary as many of the strings in lib are translated. Dselect is otherwise unchanged.

Changes:

The files in intl and po were taken from gettext 0.10.26, by way of the gettextize program. I altered the makefiles to remove the VERSION symbol, which is only used in targets which dpkg does not support.

aclocal.m4 was stolen from the textutils package, configure.in was altered to use these new tests, symbols were added to acconfig.h, and the two new directories were added to Makefile.in.

The dpkg, dpkg-deb, md5sum, split, and dselect Makefiles now look for headers in ../intl, and try to link with whatever i18n libraries configure finds. They also now define LOCALEDIR in CFLAGS.

include/dpkg.h has the necessary NLS boilerplate, and the only file which dosen't include this (md5sum/md5sum.c) had it added directly.

The most intrusive change is due to a disagreement between xgettext and the dpkg coding style. Although xgettext understands string constant concatenation, it can't handle the case where preprocessor symbols are used as well. The dpkg code uses this a lot, especially in cases like this:

```
ohshite("error reading from " BACKEND " pipe");
```

where BACKEND is defined as "dpkg-deb". Because xgettext can't handle this, I have changed this usage in all cases to something like:

```
ohshite(_("error reading from dpkg-deb pipe");
```

This isn't very kind to Ian, I know. But what can I do?

3 Code Internals

3.1 Structure Definitons

3.1.1 versionrevision

```
struct versionrevision {
  unsigned long epoch;
  char *version;
  char *revision;
};
```

The versionrevision structure is used to store a Debian version specification

3.2 Functions

3.2.1 Parsing translation tables

Each of these tables is used to associate a set of strings with a corresponding set of integers.

Current tables are:

```
const struct namevalue booleaninfos[];
```

Maps boolean strings (`"yes"`,`"no"`) to their binary values.

```
const struct namevalue priorityinfos[];
```

Maps priority strings (`"required"`, `"important"`, `"standard"`) to values of type `enum pkgpriority`.

```
const struct namevalue statusinfos[];
```

Maps package status strings to values of type `enum ???`.

Package status strings are:

"not-installed" "unpacked" "half-configured" "installed" "half-installed" "config-files" "postinst-failed" "removal-failed"

```
const struct namevalue eflaginfos[];
```

```
const struct namevalue wantinfos[];
```

```
const struct nickname nicknames[];
```

Maps obsolete control fields to their current versions.

3.2.2 Parsing functions

3.2.2.1 parseerr

```
void parseerr
(FILE *file, const char *filename, int lno,
 FILE *warnto, int *warncount, const struct pkginfo *pigp,
 int warnonly, const char *fmt, ...);
```

Report an error parsing a control data stream. Checks *file* for error status on *file*, printing an error message to stderr and exiting with error status if an error is present on the stream. Formats the variable arguments according to the `printf`-style string in *fmt*, preceding the result with *filename*, *lno*, and *pigp* (if not NULL). If *warnonly* is 0, writes the result to stderr and exits with error status. Otherwise, increments *warncount* and returns normally.

3.2.2.2 illegal_packagename

```
const char *illegal_packagename
(const char *p, const char **ep)
```

Checks the package name at *p* for proper syntax. Returns NULL in case of success, setting **ep* to point after the last character in the package name, including any trailing whitespace (as defined by isspace()). In case of error, returns the error message as a pointer to a static buffer.

3.2.2.3 informativeversion

```
int informativeversion(const struct versionrevision *version)
```

Returns true iff the `versionrevision` is non-empty; otherwise returns false.

3.2.2.4 varbufversion

```
void varbufversion
(struct varbuf *vb,
 const struct versionrevision *version,
 enum versiondisplayepochwhen vdew);
```

Writes a human-readable representation of *version* to *vb*. Possible values of *vdew*:

vdew_never
: never include epoch

vdew_nonambig
: include epoch string if non-zero

vdew_always
: always include epoch string

3.2.2.5 versiondescribe

```
const char *versiondescribe
(const struct versionrevision *version,
 enum versiondisplayepochwhen vdew);
```

Return a human-readable representation of *version* as a pointer to a static buffer. If the version structure is empty, return `"<none>"`. Static buffers are allocated from a set of 10 in round-robin fashion (they will be re-used after 10 successive calls to `versiondescribe`).

3.2.2.6 parseversion

```
const char *parseversion
(struct versionrevision *rversion, const char *string);
```

Parse the contents of *string* into *rversion*. If successful, returns NULL, otherwise returns a pointer to a static error string.

3.2.2.7 parsemustfield

```
void parsemustfield
(FILE *file, const char *filename, int lno,
 FILE *warnto, int *warncount,
 const struct pkginfo *pigp, int warnonly,
 char **value, const char *what);
```

3.2.2.8 skip_slash_dotslash

```
const char *skip_slash_dotslash (const char *p);
```

3.2.2.9 convert_string

```
static int convert_string
(const char *filename, int lno, const char *what, int otherwise,
 FILE *warnto, int *warncount, const struct pkginfo *pigp,
 const char *startp, const struct namevalue *nvip,
 const char **endpp)
```

Table of Contents

1 Copyright **1**

2 Build Notes **2**

2.1 Automake 2
2.2 Internationalization 4

3 Code Internals **6**

3.1 Structure Definitons 6
3.1.1 versionrevision 6
3.2 Functions 6
3.2.1 Parsing translation tables 6
3.2.2 Parsing functions 6
3.2.2.1 `parseerr` 7
3.2.2.2 `illegal_packagename` 7
3.2.2.3 `informativeversion` 7
3.2.2.4 `varbufversion` 7
3.2.2.5 `versiondescribe` 7
3.2.2.6 `parseversion` 8
3.2.2.7 `parsemustfield` 8
3.2.2.8 `skip_slash_dotslash` 8
3.2.2.9 `convert_string` 8

C

convert_string() 8

I

illegal_packagename() 7
informativeversion() 7

P

parseerr() 7
parsemustfield() 8
parseversion() 8

S

skip_slash_dotslash() 8

V

varbufversion() 7
versiondescribe() 7