Boueux

an open-source clone of MuddyGB
documentation for version 0.1, January 2014

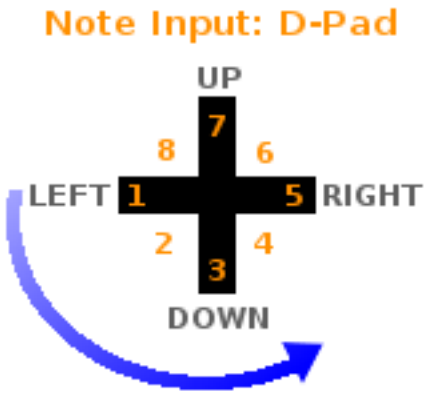
Jack Willis

1 Overview

The GNU `hello` program (http://www.gnu.org/software/hello/) produces a familiar, friendly greeting. It allows nonprogrammers to use a classic computer science tool which would otherwise be unavailable to them. Because it is protected by the GNU General Public License, users are free (in perpetuity) to share and change it.

```
(sound-on!)

(printf "uMuddy v%s\n\n" *UMUDDY-VERSION*)

(loop
    (set 'keys (joypad))

    (if (pressed? 'start)
        (begin
            (set 'octave (not octave))
            (printf "\n* octave +%d\n" octave)

            (waitpadup)
            (delay *NOTE-ACCURACY*)))

    (set 'this-pos (scale-position keys))
    (set 'this-note (get-note this-pos octave))

    (if (and this-note this-pos)
            (if (!= this-note last-note)
                    (begin
                        (sound-on!)
                        (play-note this-note)
                        (printf "%d" this-note)))
             (begin
                (sound-off!)
                (if (and last-note this-pos)
                    (printf "."))))

    (set 'last-note this-note)
```

```
(delay *NOTE-ACCURACY*))
```



Not to spoil the joke, but of course the practical purpose of GNU Hello is to serve as a minimal example of a GNU package. So, although most manuals don't need to discuss the implementation of the programs they document, that is part of the goal here.

First, GNU Hello follows the GNU coding standards (see Section "Preface" in *GNU Coding Standards*) and GNU maintainer standards (see Section "Preface" in *Information for GNU Maintainers*). These are the basic documents which all GNU packages should adhere to.

The Hello package also implements recommended development practices not embodied in the standards, using other GNU packages and features:

- It uses Automake (see Section "Introduction" in *GNU Automake*) and hence also Autoconf (see Section "Introduction" in *GNU Autoconf*) for configuration.
- It uses Gnulib (see Section "Introduction" in *GNU Gnulib*) to enhance portability and avoid duplication of common sources. Both `gnulib-tool` and `srclist-update` are used, for purposes of example. See the '`README-dev`' file in the distribution.
- GNU Gettext (see Section "Introduction" in *GNU Gettext*) is used for internationalization support. Hello's greeting has been translated into many languages.
- Internally, Hello uses the GNU `getopt_long` function (see Section "Getopt Long Options" in *GNU C Library*) to parse options, thus supporting GNU-style long options such as '`--help`'.
- The Hello Man page is generated with GNU `help2man` (see Section "Overview" in *GNU help2man*) from the '`--help`' output. This relieves the maintainers from the burden of updating separate man documentation, yet provides a reasonable overview for man devotees.
- Finally, Texinfo (see Section "Introduction" in *Texinfo*) is the documentation format for this manual. It supports output in Info, HTML, PDF, DVI, plain text, XML, and other formats.

GNU Hello is implemented in C. The GNU Gettext distribution contains "hello world" examples in many other programming languages; see the Gettext home page at `http://www.gnu.org/software/gettext/`.

The top-level '`Makefile.am`' in Hello also contains a few special targets for other projects to adapt as desired:

`diff` Make a diff from the previous release, assuming the current tarball is in the current tarball.

`po-check` Verify that all source files using `_()` are included for translation in '`po/POTFILES.in`', so translators will have all the messages.

`wwwdoc` Sample procedure for updating the manual on the GNU web site, in this case `http://www.gnu.org/software/hello/manual/`.

2 Sample output

Here are some examples of running GNU Hello.

This is the output of the command '`hello`':

```
Hello, world!
```

This is the output of the command '`hello --traditional`':

```
hello, world
```

This is the output of the command '`hello --greeting=hi`':

```
hi
```

3 Invoking `hello`

The format for running the `hello` program is:

```
hello option ...
```

With no options, `hello` prints the greeting '`Hello, world!`'.

`hello` supports the following options:

'`--greeting=`*text*'
'`-g` *text*' Output *text* instead of the default greeting.

'`--help`'
'`-h`' Print an informative help message on standard output and exit successfully.

For the '`--help`' output of GNU programs, it's strongly encouraged to include a brief (one or two sentences) description of what the program does, as well as the synopsis of how to run the program. Any environment variables which affect execution should also be mentioned (Hello doesn't have any).

'`--next-generation`'
'`-n`' Output the greeting, but possibly including box-drawing characters or other fancy stuff, especially in translated locales. (If you would like to volunteer to translate messages for GNU packages, please see http://translationproject.org.)

'`--traditional`'
'`-t`' Output the traditional greeting message '`hello, world`'.

'`--version`'
'`-v`' Print the version number and licensing information of Hello on standard output and then exit successfully.

If more than one of the greeting options ('`-g`', '`-n`', '`-t`', and their long-named equivalents) is specified, whichever comes last takes precedence.

4 Reporting bugs

To report bugs, suggest enhancements or otherwise discuss GNU Hello, please send electronic mail to `bug-hello@gnu.org`.

For bug reports, please include enough information for the maintainers to reproduce the problem. Generally speaking, that means:

- The version numbers of Hello (which you can find by running '`hello --version`') and any other program(s) or manual(s) involved.
- Hardware and operating system names and versions.
- The contents of any input files necessary to reproduce the bug.
- The expected behavior and/or output.
- A description of the problem and samples of any erroneous output.
- Options you gave to `configure` other than specifying installation directories.
- Anything else that you think would be helpful.

When in doubt whether something is needed or not, include it. It's better to include too much than to leave out something important.

Patches are welcome; if possible, please make them with '`diff -c`' (see Section "Overview" in *Comparing and Merging Files*) and include '`ChangeLog`' entries (see Section "Change Log" in *The GNU Emacs Manual*). Please follow the existing coding style.

Concept index