

Building DICE Packages

Paul Anderson
Division of Informatics
University of Edinburgh
<paul@dcs.ed.ac.uk>

Summary

- Background
 - DICE Software Guidelines
 - DICE buildtools
 - CVS
 - RPM
- Building a Module with dice-buildtools
 - Creating a CVS Module
 - Using dice-example as a template
 - Variable substitution
 - Editing and committing files
 - Making RPMs
 - Signing and submitting RPMs

Dice Guidelines

- The DICE software guidelines contain rules for:
 - Package names and version numbers
 - Standard pathnames (`dice.mk` and `lcfg.mk`)
 - RPM Groups
 - Documentation formats
 - LCFG-specific packaging (default. etc.)
- Some of these standards are expected by automatic tools (eg. Web generation)
- The standards are open to suggestions for evolution!
- Documentation
 - <http://www.dice.informatics.ed.ac.uk/doc/dice-guidelines.pdf>
 - `/usr/lib/dice/doc/pdf/dice-guidelines.pdf`

Build Tools

- An optional file which can be included in the package Makefile to provide support for:
 - Tagging and incrementing *CVS* versions according to the standards
 - Substituting configuration variables in text files
 - Creating RPMs from the *CVS* or working files
- This has the advantages:
 - Simple starting point for those with little *CVS* or RPM knowledge
 - Provides configurable source files with little work
 - A common standard that more people will understand
- Documentation
 - <http://www.dice.informatics.ed.ac.uk/doc/dice-buildtools.pdf>
 - `/usr/lib/dice/doc/pdf/dice-buildtools.pdf`

CVS Preparation

- Set ssh keys to access CVS without giving a password. See Chris' documentation:
 - http://www.dcs.ed.ac.uk/home/cc/dice_cvs.pdf
- Set up environment for the DICE CVS server:
 - `CVS_RSH=ssh`
 - `CVSROOT=$USER@cvs.dcs.ed.ac.uk:/cvs/dice`

RPM Preparation

- Create a `~/ .rpmmacros` file to define the location where you want to build RPMs:

```
%_packager Paul Anderson <paul@dcs.ed.ac.uk>
```

```
%_gpg_name Paul Anderson <paul@dcs.ed.ac.uk>
```

```
%_signature gpg
```

```
%_sourcedir %(echo $RPMROOT)/SOURCES
```

```
%_rpmdir %(echo $RPMROOT)/RPMS
```

```
%_specdir %(echo $RPMROOT)/SPECS
```

```
%_srcrpmdir %(echo $RPMROOT)/SRPMS
```

```
%_builddir %(echo $RPMROOT)/BUILD
```

- This version is parameterized on the environment variable which lets you have different locations

- Also strongly recommended that you have a *GPG* key:

- `Man gpg`

Creating a New Module

- Create a directory on the *CVS* server:
 - `ssh cvs`
 - `mkdir /cvs/dice/FOO`
 - `chmod 02775 /cvs/dice/FOO`
- You can manually copy in any existing RCS ,v files
- Go to a working directory and check the module out:
 - `cvs checkout FOO`
- Create files in FOO
- Add each file (or directory)
 - `cvs add my-files`
- Either use *dice-buildtools*, or read the *CVS* and *RPM* manuals.

DICE Example

- Checkout the dice-example module:
 - Cvs checkout dice-example
- It contains:
 - `myprog.cin` (your program)
 - `myprog.pod.cin` (your docs)
 - `README` (your readme)
 - `config.mk` (the key thing!)
 - `Makefile` (no change until you need to)
 - `specfile` (no change until you need to)
 - `README.BUILD` (no change)
 - `ChangeLog` (create an empty one)
- You can copy these files into your own module as a starting point (remember to cvs add them)

The config.mk file

- Contains configuration variables
- Some are used by dice-buildtools, most are just user-defined:

```
NAME=dice-example
```

```
DESCR=An Example DICE program
```

```
V=1.0.4
```

```
R=1
```

```
GROUP=DICE/Utilities
```

```
AUTHOR=Paul Anderson <paul@dcs.ed.ac.uk>
```

```
PROG=myprog
```

```
MANSECT=1
```

```
MANDIR=$(DICEMAN)/man$(MANSECT)
```

```
DATE=13/03/02 14:53
```

The Program

- The program can contain configuration variables, which are substituted at "make" time:

```
#!/bin/sh
```

```
echo "Hello World"
```

```
echo "Program version @V@ by @AUTHOR@"
```

```
exit 0
```

The Documentation

- The documentation is in POD format
 - Buildtools can make man pages very easily from this
 - HTML can be easily generated
 - It is very easy to write
 - Man perlpod for details
 - Variable gets substituted here too

- `Myprog.pod.cin`

```
=head1 NAME
```

```
example - An example DICE program
```

```
=head1 DESCRIPTION
```

```
This program is an example only.
```

```
=head1 OPTIONS
```

```
=over 4
```

```
=item B<-n>
```

```
This option is an example only and doesn't do  
anything.
```

```
=back
```

Variable Substitution

- Buildtools uses a Makefile rule to create any file from the corresponding .cin file by performing variable substitution:
 - The example Makefile creates the program and the documentation as default targets (just type "make")
- Variables have the form @NAME@
- The Makefile target `confi.g.tex` will create a file of TeX definitions for the variables
- All the variables from `dice.mk` and `lcfg.mk` are also included by default
- The file `dice-test.mk` can contain values to override other values, only during testing

Releases and Versions

- Buildtools automatically manages the version number in the config.mk file, checks the current files into CVS, and tags the CVS files according to the standards
- There is a makefile target for each component of the version number (x.y.z):
 - Make `release / minorversion / majorversion`
- Before using any of these to check in a new version, create a ChangeLog entry:
 - Use `m-x add-changelog-entry` in Emacs
- The module can be checked out at any time with:
 - `cvs checkout -r FOO_x_y_z`

Making RPMs

- Buildtools has targets for creating an RPM either from the latest version in the CVS, or the working files:
 - `make rpm / devrpm`
- Development RPMS made from the working copy are tagged with `_dev` on the name and should not be submitted to the repository
- Tarballs without RPMs can be created with:
 - `make pack / devpack`

Submitting RPMs

- RPMs should be PGP signed before submission. You need to locate both the SRPM and all the corresponding RPMs in the directories specified in the `~/ .rpmmacros` file:
 - `rpm -resign rpm-filenames`
- Submit the binary RPMs to the repository (the source RPMs will automatically be located and submitted):
 - `rpmsubmit rpm-filenames`
- Adding the RPMs to the appropriate LCFG header files with LCFG will cause them to be installed on the corresponding machines. Eg:
 - `rfe rpm71/local.h`

Miscellaneous

- See the `lcfg-example` module for a slightly more complex example
- The `prep` and `devprep` targets can be used to perform extra processing when packing the tarball.
- The `PROD` and `DEV` variables can be used to include different files at development and production time