DICE and LCFG Software Guidelines

by Paul Anderson <paul@dcs.ed.ac.uk>

DICE Computing Environment Project
Division of Informatics
University of Edinburgh

1 Introduction

This document describes the standards to be used for system software produced within the Division of Informatics. This is intended to be an evolving document, based on experiences and recommendations from all Divisional COs, and suggestions for changes and additions are always welcome.

It is hoped that these guidelines will help to make Division software easier to work with, more reliable, and accessible to more people. In particular, all local software in use on production systems must have a corresponding source module in the DICE CVS repository, including up-to-date documentation, and adequate instructions for regenerating the software from the sources.

2 CVS and Packages Names

The CVS repository has a flat structure with an entry at the top level for each module. Each module has a unique name and normally corresponds to one SRPM of the same name (the SRPM may generate several RPMS).

`lcfg` modules are intended to be exportable as part of the LCFG configuration system, and should not contain any site-specific code. LCFG component (object) modules should have names of the form:

`lcfg-`*component*

although the module name should not be used to determine which RPMs contain components (this can be determined by the RPM group - see section 6).

Copies of imported packages which have significant local modifications should be named:

`dice-`*package*

There are no other restrictions on module names.

3 Version Numbering

Module version numbers should consist of three components (X.Y.Z), interpreted as follows:

- X is a major version number which should be changed when there are substantial changes to the code base or interfaces.
- Y is a minor version number. Odd numbers represent a *development* version of the package, and even numbers represent a *production* version.
- For production version, Z represents a bug-fix level. There should be normally be no significant functional difference between different bug-fix levels of the same version, and they should always be be backwards-compatible.
- For development versions, Z represents a development release number. In this case, there may well be significant differences in functionality between different releases.
- By convention, Y=99 represents the preparation for the next major version. This means that a new package should be developed at 0.99.Z, and move to `1.0.0` for the first production release.

CVS tags should be used to identify all files corresponding to a particular version of the package. The tags should have the form:

*name*_X_Y_Z

The *name* is the name of the module with any "-"s replaced with "_"s.

Notice that the RPM release number is not part of this versioning scheme. The RPM release number will normally always be 1, but it may increase if the packaging is changed without changing the contents of the CVS repository (for example, if a specfile which is not stored in the CVS is changed).

4 Pathnames

The packages `dice-config` and `lcfg-config` provide symbolic names for the common pathnames

used by DICE and LCFG software. The files `dice.mk` and `lcfg.mk` can be included using GNU `make` as follows:

```
include dice.mk
include lcfg.mk
```

Appendices A and B list the symbols currently provided and describe their use. Additional suggestions for these files are welcome.

All packages should configure themselves at build time to use these pathnames wherever appropriate; this allows the pathnames to be changed by a simple rebuild, either for future evolution (maybe on different platforms), or by different sites (who might supply a different `lcfg-config` package). Notice that the example files given in appendices C and E contain explicit pathnames; these would normally be instantiated at build time using a package such as `dice-buildtools` (see 5) or GNU `autoconf`.

The file `lcfg.mk` will define legacy values for LCFG pathnames if the environment variable `LCFG_COMPAT` is set to `yes`, and new pathnames if it is set to `no`. At the time of writing, the DICE LCFG system is still using these legacy pathnames and the *default is to use legacy pathnames*. Other filenames should conform to the Linux Filesystem Hierarchy Standard (FHS)[1] wherever possible.

5 Building and Packaging

The package `dice-buildtools`[2] provides a set of makefile targets to assist with building and packaging DICE software from the CVS repository. Use of `dice-buildtools` is recommended, since this helps to ensure conformance to the standards, and provides a common framework which will be familiar to other users. Feedback and suggestions for improvements to this process are welcome.

Whether or not `dice-buildtools` is used, the following files must appear in the CVS and the `%doc` section of any corresponding RPM:

- `README` – a short description of the module for users.
- `README.BUILD` – instructions for packaging and installing the module from the CVS sources. In particular, it must be clear how to re-configure the package and generate the the corresponding RPMs.
- `ChangeLog` – A history of modifications. This should include a brief description of changes between versions. The Emacs command `M-x add-change-log-entry` will automatically add a correctly-formatted entry, and `cvs2cl` will create a ChangeLog from the comments in the CVS repository.

[1] http://www.pathname.com/fhs/
[2] http://www.dice.informatics.ed.ac.uk/doc/dice-buildtools.pdf

6 RPMs

Generated RPMs should include some standard headers:

```
Vendor: University of Edinburgh
Copyright: GPL
Group: GROUP (see below)
Packager: name <mail>
```

The `Distribution` field is reserved for a future standard use and should not be assigned.

The following groups should be used where appropriate:

- `LCFG/Components` - LCFG components.
- `LCFG/Doc` - LCFG documentation.
- `LCFG/Defaults` - LCFG default resources.
- `LCFG/System` - Other LCFG-related system software.
- `DICE/Applications` - DICE (user) applications.
- `DICE/System` - DICE system software.
- `DICE/Config` - DICE local configuration files.
- `DICE/Doc` - DICE documentation.

The RPM must contain an accurate `%changelog` section. The utility `cl2rpm` can be used to generate this automatically from a GNU-format `ChangeLog` file. (`dice-buildtools` performs this automatically).

Care should be taken to ensure that RPMs do not depend unnecessarily on the local environment. Some common problems include:

- Files should not be owned by local users; use `%defattr` (see the example SPEC file in appendix C).

- ❑ The RPM `BuildRoot` should not refer to some local directory (The value shown in the sample SPEC file works well).

- ❑ If documentation compilation requires specific local tools (such as LaTeX styles), then it is useful to include a copy of the compiled documentation in the SRPM. This allows the RPM to be recompiled at a remote site, without rebuilding the documentation.

7 LCFG components

In addition to the normal code/documentation RPM, LCFG component modules should generate a separate RPM for the default resources. This is necessary so that the default resources can be collated on separate servers. This is not difficult, and Appendix C shows a sample RPM SPEC file which can be used as a template.

7.1 Component Code

The document *Writing LCFG Components*[3] provides a detailed description of the semantics, conventions and tools for creation of LCFG components.

The LCFG component code should be packaged in an RPM with the name `lcfg-`*component*. This should include the LCFG component itself and any small supporting programs or other files. If there are significant related programs which may be optional, these can be packaged as separate RPMS; for example, `lcfg-foo-server` and `lcfg-foo-client` (as well as `lcfg-foo` for the component itself).

The component code should not contain any site-specific configuration files. These should be supplied by separate modules. In many cases, the component RPM is not architecture-dependent and should have the `BuildArch` set to `noarch`.

The component should use the directories specified in `lcfg.mk` (see Appendix B).

7.2 Component Documentation

The original LCFG `doc` method is now deprecated and should not be supported. However, each component module *must* include a documentation file in POD (plain-old-documentation) format [4], listing the supported methods and resources, together with a brief description of the component. This format is very easy to create, and can easily be converted into other formats such as HTML, manual pages, or TeX. Appendix D shows a sample POD file.

The POD file should be installed in

```
$(LCFGPOD)/component.pod
```

The component module must also contain a manual page, which can easily be generated automatically from the POD file:

```
pod2man '--section=8'
        '--release=Release: ver'
        '--center=LCFG/Components'
        foo.pod >lcfg-foo.8
```

Notice the standard headings, and also the fact that the manual page name includes the `lcfg-` prefix. The manual page should be included in the same RPM as the component source for installation in section 8 of the manual pages on the local machine.

7.3 Default Resources

Each component module must include a file with the name *component*`-`*schema*`.def` listing the supported resources, together with appropriate default values and meta-resources (see the document *LCFG Adaptors for XML Profiles*[5]). The defaults file must be site-independent, so it should not include any site-specific information; site-specific default values should be provided by a site header file.

The *schema* is a schema version number which should be incremented whenever the resource schema is changed – including this as part of the filename allows a single server to support clients with different component versions. A resource with the name `schema` should also be defined with the schema version as the default value. This allows components to check for mismatches in schema versions.

The defaults file should be included in the component RPM and installed in

```
$(LCFGCLIENTDEF)/component-schema.def
```

An LCFG profile server needs access to the default files for all the clients which it serves. This may include serving multiple schema versions. For this reason, each module should also generate a separate

[3] http://www.lcfg.org/doc/lcfg-ngeneric.pdf
[4] `perldoc perlpod`
[5] http://www.lcfg.org/doc/lcfg-profile.pdf

RPM with the name `lcfg-`*component*`-defaults-s`*schema*, containing only a copy of the defaults file, installed in

```
$(LCFGSERVERDEF)/component-schema.def
```

Using a relocatable RPM for the defaults file is straightforward and helpful for sites which store their sources in a different location (see the example SPEC file in Appendix C).

8 Documentation

Some documentation will be included with the corresponding code module. Other, more general, documentation may be managed as a separate module. Documentation files should be installed in the directories `$(`*TYPEFORMAT*`)` where *TYPE* is either `DICE` or `LCFG` and *FORMAT* is:

- `PDF` – PDF files named as *module*`.pdf`
- `HTML` – subdirectories with the same names as the modules, containing an `index.html` file together with any other referenced HTML files.
- `BIB` – latex-style bib files (with the name *module*`.bib`) containing meta-information for the documents. These can be used to generate indices, or for inclusion in bibliographies. See Appendix E for an example.
- `POD` – POD files. It is useful to install POD versions of manual pages in this directory so that they can be published on the web server.

9 Importing and Modifying External Packages

If an external package is imported and modified sufficiently that it becomes a separate "fork" of the original code, then that package can be treated just like any other local package, with full source code in the CVS repository, and a new local version numbering scheme. It is recommended the the package be named `dice-`*package* so that the original provenance is clear. It is up to the person modifying the package to decide whether the RPM reflects this by including the original tarball and local patches, or whether it simply contains the final local code.

There are currently no good recommendations for managing small changes to external packages, especially if the external code base is large. By convention, local RPMS are created with a postfix on the RPM revision. For example:

```
foobar-1.2.b03-12
```

Becomes:

```
foobar-1.2.b03-12.dice.3
```

This ensures that locally modified versions take precedence over the original copies, but it preserves the original name and version for dependency relationships. However, it means that different release numbers of the RPM contain different code (not just packaging differences), and there is no easy way of tracking the development of the local changes in the CVS repository.

Better suggestions for handling this are welcome.

Appendix A DICE Symbols Defined in dice.mk

DICEBIB	Directory for BIB files.
DICEBIBURL	URL for BIB files.
DICEBIN	Directory for user binaries.
DICEDOC	Base directory for documentation.
DICEHTML	Directory for HTML files.
DICEHTMLURL	URL for HTML files
DICELIB	Base directory for read-only files.
DICEMAN	Directory for man pages.
DICEPDF	Directory for PDF files.
DICEPDFURL	URL for PDF files.
DICEPERL	Directory for Perl modules. Normally in the subdirectory DICE::.
DICEPOD	Directory for POD files.
DICESBIN	Directory for system binaries.
DICEURL	Base URL for documentation

Based on version 1.0.4 of dice-config.

Appendix B LCFG Symbols Defined in lcfg.mk

LCFGBIB	Directory for BIB files.
LCFGBIBURL	URL for BIB files.
LCFGBIN	Directory for user binaries.
LCFGCLIENTDEF	Directory for default resource files used by client
LCFGCONF	Directory for generated files to be preserved between object runs. Files are normally prefixed with the module name, or stored in subdirectories with the same name as the module.
LCFGCONFIGMSG	A message stating whether the configuration variables have been set to old-style compatibility values, or new-style values.
LCFGDATA	Directory for templates and other fixed configuration files. Files are normally prefixed with the module name, or stored in subdirectories with the same name as the module.
LCFGDEF	Deprecated (use LCFGSERVERDEF)
LCFGDOC	Base directory for documentation.
LCFGHTML	Directory for HTML files.
LCFGHTMLURL	URL for HTML files.
LCFGLIB	Base directory for read-only files.
LCFGLOCK	Directory for lock files.
LCFGLOG	Directory for log files.
LCFGMAN	Base directory for man pages.
LCFGPDF	Directory for PDF files.
LCFGPDFURL	URL for PDF files.
LCFGPERL	Directory for Perl modules. Normally in the subdirectory `LCFG::`.
LCFGPOD	Directory for POD files.
LCFGROTATED	Directory for log rotate files.
LCFGSBIN	Directory for system binaries.
LCFGSERVERDEF	Directory for default resource files used by server
LCFGSTATUS	Directory for status files.
LCFGTMP	Directory for temporary files (may be deleted when objects are not running). Files are normally prefixed with the module name, or stored in subdirectories with the same name as the module. Components should not store temporary files in system tmp directories.
LCFGURL	Base URL for documentation.

Based on version 1.0.7 of `lcfg-config`.

Appendix C Sample RPM SPEC File for an LCFG Component

```
Summary: An Example LCFG component
Name: lcfg-example
Version: 1.0.10
Vendor: University of Edinburgh
Release: 1
Copyright: GPL
Group: LCFG/Components
Source: lcfg-example-1.0.10.src.tgz
BuildArch: noarch
BuildRoot: /var/tmp/%{name}-build
Packager: Paul Anderson <paul@dcs.ed.ac.uk>
Requires: lcfg-ngeneric

%description
An example LCFG component.
(configured with old-style filenames for compatibility)

%prep
%setup

%build
make

%install
rm -rf $RPM_BUILD_ROOT
make PREFIX=$RPM_BUILD_ROOT install

%postun
[ $1 = 0 ] && rm -f /etc/logrotate.d/lcfg-example
exit 0

%files
%defattr(-,root,root)
%doc ChangeLog README README.BUILD
%doc /usr/man/man8/*
%doc /usr/lib/lcfg/doc/pod/example.pod
/etc/obj/example
/usr/lib/lcfg/conf/*
/var/obj/conf/example/
/usr/lib/lcfg/defaults/client/example-1.def
# These files are only included because we want to include the
# source files as documentation since this is an example
%doc Makefile specfile example.cin example.pod config.mk

%package defaults-s1
Summary: Default resources for lcfg-example
Group: LCFG/Defaults
Prefix: /usr/lib/lcfg/defaults/server
BuildArch: noarch
%description defaults-s1
Default resources for the LCFG example component.
```

```
(configured with old-style filenames for compatibility)
%files defaults-s1
%defattr(-,root,root)
/usr/lib/lcfg/defaults/server/example-1.def

%clean
rm -rf $RPM_BUILD_ROOT
```

Appendix D Sample LCFG Component POD File

```
=head1 NAME

example - An example LCFG component

=head1 DESCRIPTION

This component is an example only.

=head1 RESOURCES

=over 4

=item B<server>

An example resource which gets substituted into the configuration file.

=back
```

Appendix E Sample BIB File

```
@miscellaneous        { dice-guidelines,
author              = { Paul Anderson <paul@dcs.ed.ac.uk> },
organization        = { University of Edinburgh },
title               = { DICE and LCFG Software Guidelines },
year                = { 2001 },
keywords            = { dice cvs },
howpublished        = { Internal Document },
file                = { /usr/lib/dice/doc/pdf/dice-guidelines.pdf },
url                 = { http:// ... /dice-guidelines.pdf }
}
```