Writing LCFG Components

Paul Anderson <*dcspaul@inf.ed.ac.uk*>

http://homepages.inf.ed.ac.uk/dcspaul

LCFG Tutorial, June 2007

Overview

- An example daemon
  - **`chatterd`**
- The LCFG component framework
  - **`lcfg-skeleton & buildtools`**
- Creating the schema file
  - **`chatter.def`**
- Creating the component
  - **`chatter`**
- Some reflections

Chatterd

```
#!/bin/sh

# The message comes from the command line argument
message=$1

# Save the PID of the daemon so we can find it
echo $$ >/var/lcfg/tmp/chatterd.pid

# Log the fact that we are starting
echo `date` : chatterd starting >>/var/log/chatterd

# Chatter away - write message to log every 2 seconds
while true ; do
  echo `date` : $message >>/var/log/chatterd
  sleep 2
done
```

An example daemon

- **`/usr/lib/chatterd`** *message*
  - writes the *message* to the log every 2 seconds
  - writes the PID to a file so we can find it
- Run it (as root)
  - **`/usr/lib/chatterd "some message"`**
- Watch the log (in another window)
  - **`tail -f /var/log/chatterd`**
- Stop the daemon (as root, in another window)
  - **`kill `cat /var/lcfg/tmp/chatterd.pid`**
- Managing this requires a custom component
  - we want to restart when the resources change

Overview

☑ An example daemon

- The framework
- The schema file
- The component
- Some reflections

We now have a simple daemon.

We are going to look at the LCFG framework so we can use it to create a controlling component

The component framework

- We need to create:
  - a *schema file*, containing resource type definitions, default values & validation code
  - the *component* code
- LCFG provides some support for this:
  - **`lcfg-ngeneric`** provides default code for most of the component with methods which can be overriden
  - **`lcfg-buildtools`** provides makefiles for configuring, building packages, installing, etc.
  - **`lcfg-skeleton`** generates skeleton files for a new component

Creating a new component

```
[lcfgfc6@localhost ~] lcfg-skeleton
Name of component [mycomp] ? chatter
One line description [] ? Example component
Add to CVS (yes/no) [no] ? no
Perl (pl) or Shell (sh) [sh] ? sh
Component author [] ? Joe Smith
Author email [lcfgfc6@localdomain] ? Joe@foo.com
Platforms [Redhat9, ..., Fedora5] ? Fedora5
Include regression test files (yes/no) [yes] ? no
Restart component on RPM update (yes/no) [yes] ? yes
file: ChangeLog
.....
lcfg-chatter not added to cvs
[lcfgfc6@localhost ~]
```

Component files

- We will not be using these files -
  - **`Changelog`** - CVS changelog
  - **`README`** - the component readme
  - **`README.BUILD`** - generic readme
  - **`specfile`** - used to build an RPM automatically
    - this requires a CVS repository
  - **`chatter.pod.cin`** - documentation skeleton
- You must delete this file (but not the others) -
  - **`test.mk`** - specific resource values for testing

Component files

- These are the files we will use -
  - **`chatter.def.cin`** - skeleton for the schema
  - **`chatter.cin`** - skeleton for the component
  - **`config.mk`** - build-time configuration variables
- The variables in config.mk are substituted in the **`.cin`** files by the **`buildtools`** makefiles -
  - type **`make`**
  - compare **`chatter.cin`** and **`chatter`**
  - type **`make clean`**

Config.mk

```
COMP=chatter
NAME=lcfg-$(COMP)
DESCR=Example component to control daemon
V=0.99.0
R=1
SCHEMA=1
VERSION=$(V)
GROUP=LCFG
AUTHOR=Paul Anderson <dcspaul@inf.ed.ac.uk>
PLATFORMS=Fedora5

MANDIR=$(LCFGMAN)/man$(MANSECT)

DATE=30/05/07 10:48
```

Overview

- ☑ An example daemon
- ☑ The framework
- The schema file
- The component
- Some reflections

We have used the LCFG framework to create skeletons for the schema and the component code.

We are now going to fill in the schema skeleton to define the resources we need for the component.

Creating the schema file

- Edit the schema skeleton **`chatter.def.cin`** to include a new resource:

```
#include "ngeneric-1.def"
#include "om-1.def"
schema @SCHEMA@
message undefined
@message %string(message): !/^undefined$/
```

- The first new line defines a new resource **`message`** with the default value **`"undefined"`**
- The second new line is a validation which says that the final value of the resource *must not* be the string **`"undefined"`**

Using the schema

- Type **`make install`** to install the schema file
  - this goes in to **`/usr/lib/lcfg/defaults`**
- Edit the **`localhost`** profile to add the new component

```
!profile.components mADD(chatter)
profile.version_chatter 1
```

- The server will notice the change to the **`localhost`** profile and recompile it
- Check the status page
  - why does the profile show an error?

Mutation (an aside)

- The compiler will generate an error if a resource is redefined -
  - **`foo.bar some value`**
  - **`foo.bar some other value`**
- To change a defined value, the compiler supports *mutations* which allow you to specify how the old and new values are combined -
  - **`foo.bar some value`**
  - **`!foo.bar mSET(some new value)`**
  - **`!foo.bar mADD(a string to the end)`**
- See the LCFG guide (5.2.4)

Configuring the host

- You might want to watch the server and client logs (in separate windows) while we do this ..
  - `tail -f /var/lcfg/log/server`
  - `tail -f /var/lcfg/log/client`
- Edit the `localhost` profile to add a value for our new resource

  `chatter.message Hello World`
- The server will compile it again
  - check the status page & watch the logs
- The client should now have the new resources
  - `qxprof chatter`

Overview

- ☑ An example daemon
- ☑ The framework
- ☑ The schema file
- The component
- Some reflections

We have created a schema file for the new component and added the schema to the host, and defined values for the resources.

We are now going to implement a component to process these resources.

What is a component ?

- A *component* is a simple script which is called by the LCFG framework with various *methods* -
  - **`/usr/lib/lcfg/components/foo start`**
- The methods are usually called using the **`om`** command, which can also be issued manually -
  - **`om foo start`**
  - **`om foo stop`**
  - **`om foo configure`**
- The framework provides a library for the component to access the resource values, and default code for all the methods

The component skeleton

```
#!@SHELL@

@TESTSHELLV@ . @LCFGCOMP@/ngeneric

Configure() {
  # Set up configuration here
  return
}

Start() {
  # Start daemon here
  return
}

Stop() {
  # Stop daemon here
  return
}

Dispatch "$@"
```

- Look at the skeleton file **`chatter.cin`**

The component skeleton

```
#!@SHELL@

@TESTSHELLV@ . @LCFGCOMP@/ngeneric

Configure() {
  # Set up configuration here
  return
}

Start() {
  # Start daemon here
  return
}

Stop() {
  # Stop daemon here
  return
}

Dispatch "$@"
```

- The second line includes the **`ngeneric`** library which provides utility functions and defaults all the methods

The component skeleton

```
#!@SHELL@

@TESTSHELLV@ . @LCFGCOMP@/ngeneric

Configure() {
  # Set up configuration here
  return
}

Start() {
  # Start daemon here
  return
}

Stop() {
  # Stop daemon here
  return
}

Dispatch "$@"
```

- The second line includes the **`ngeneric`** library which provides utility functions and defaults all the methods
- The **`Dispatch`** routine processes the command line and calls the methods

The component skeleton

```
#!@SHELL@

@TESTSHELLV@ . @LCFGCOMP@/ngeneric

Configure() {
  # Set up configuration here
  return
}

Start() {
  # Start daemon here
  return
}

Stop() {
  # Stop daemon here
  return
}

Dispatch "$@"
```

- The second line includes the **`ngeneric`** library which provides utility functions and defaults all the methods
- The **`Dispatch`** routine processes the command line and calls the methods
- We need to edit the skeleton methods ...

The chatter component

```
Start() {
  Daemon "/usr/lib/chatterd '$LCFG_chatter_message'"
  return
}

Stop() {
  PID=`cat /var/lcfg/tmp/chatterd.pid`
  kill $PID
  return
}

Configure() {
  IsStarted && Stop && Start
  return
}
```

Framework support

- The framework provides some utility functions, including -
  - **`IsStarted`** - true if the component has been started
  - **`Daemon`** - start a process in the background with appropriate process group fiddling etc.
- The framework makes all the resources available as simple shell variables -
  - **`$LCFG`**_*component*_*resource*
  - Eg. **`$LCFG_chatter_message`**

Using the component

- Add the code to the component skeleton
- Do **`make install`** to install it
- Do **`om chatter start`** to start it
- Check the log file to see that it is logging the message
- Change the resource value in the profile and watch the component restart the daemon and start logging the new message. You may want to watch the logs -
  - **`tail -f /var/lcfg/log/server`**
  - **`tail -f /var/lcfg/log/client`**
  - **`tail -f /var/lcfg/log/chatterd`**
  - **`tail -f /varlog/chatterd`**

Overview

- ☑ An example daemon
- ☑ The framework
- ☑ The schema file
- ☑ The component
- Some reflections

We have created a new component to control our daemon, including the code and the schema file.

We have configured a host to use this component and defined values for the resources

We are going to finish by reflecting on the implications of all this

Some reflections

- Think about how little we wrote (<10 lines?)
- Notice that the resource values could be included in a header file, and hundreds of machines would reconfigure as soon as the header was changed
- The values in the header file are very simple, and could be auto-generated (by some autonomic process?)
- Simple conditionals in the header file can be used to make site-wide relationships
  - although *spanning maps* provide more support for this

Configuring relationships

```
#define SERVER hostA

#ifeq HOSTNAME SERVER

chatter.message I am the server
/* And a lot of other stuff to
   set up the server */

#else

chatter.message The server is SERVER
/* And stuff to set me up as a client
   of SERVER */

#endif
```

Some exercises

- Modify the **`chatterd`** component so that the message frequency can also be specified as a resource
- Look at the LCFG guide (10.3.12) and include some more robust error checking in the component.
- Assume that **`chatterd`** was more complex and required a configuration file constructing from the the resources. Use the template processor to create a config file including resource values (10.3.11)

References

- Slides for this talk:
  http://homepages.inf.ed.ac.uk/dcspaul/publications/ukuug2007b.pdf
- The Complete Guide to LCFG
  http://www.lcfg.org/doc/guide.pdf
- Paul Anderson
  http://homepages.inf.ed.ac.uk/dcspaul
  <*dcspaul@inf.ed.ac.uk*>