# The Wonderful World of Debconf

Matthew Palmer

`mpalmer@debian.org`

## Overview

- Why use Debconf?

## Overview

- Why use Debconf?
- Debconf Concepts

## Overview

- Why use Debconf?
- Debconf Concepts
- Programming Interfaces

## Overview

- Why use Debconf?
- Debconf Concepts
- Programming Interfaces
- A Packaging Example

## Overview

- Why use Debconf?
- Debconf Concepts
- Programming Interfaces
- A Packaging Example
- Debconf Host Configuration

## Overview

- Why use Debconf?
- Debconf Concepts
- Programming Interfaces
- A Packaging Example
- Debconf Host Configuration
- Deep Hacking

## The purpose of Debconf

- A unified configuration interface

## The purpose of Debconf

- A unified configuration interface
- Easy means to store configuration data

## The purpose of Debconf

- A unified configuration interface
- Easy means to store configuration data
- Simple, front-end independent, config scripts

## The purpose of Debconf

- A unified configuration interface
- Easy means to store configuration data
- Simple, front-end independent, config scripts
- Fully supports localization

## Debconf Concepts

- Structure
- Questions
- Data Storage

## Package Structure

Three things are needed for a Debconfed package - a config script, a list of questions (called a *Template*), and maintainer scripts (`postinst`, `prerm`, etc) which ask the Debconf database for config information.

The Debhelper script `dh_installdebconf` is useful for installing the template and config script. We will discuss the maintainer scripts later on.

Your package must Depend: on debconf (or Pre-Depend: if you use it in a `preinst` script) and you must get the version right for the capabilities you use in your scripts.

## Script Structure

The config script (and, to a lesser degree, the maintainer scripts) are a middle layer between the user interface and the config database. Calls to both the UI and backend are abstracted, so you never, ever need to worry whether the user is using a plaintext interface, or a web browser, or whatever. Similarly, there is no need to consider if the question data is coming from a text file or an LDAP directory.

Config scripts merely ask questions of users and make decisions for other questions based on the responses.

## Questions and Why They’re Cool

There is one fundamental datum in Debconf - the question. The question identifier ties all the pertinent information about the question together - what the question is, it’s description, and it’s current value. Everything else in the Debconf system is basically centered around manipulating these questions in various interesting fashions.

A question’s identifier is an entry in a hierarchical tree. The ‘path’ to a question is separated by slashes, like a Unix path. An example would be `foo/bar/myquestion`.

## Data Storage

Question data (both the questions themselves and their current answers) is stored in a database (or databases) somewhere. The mechanics of these databases, and how to select the format to use, is the topic of the final parts of this talk. At this time, it is sufficient to say that there is a generic mechanism for storing questions and their answers, which is accessed via some debconf calls. Typically the data will be accessed via the question's identifier (ie `foo/bar/myquestion`).

## Templates

These provide the question data, such as what sort of answer we want, and what the question is about. For example:

```
Template: foo/bar/myquestion
Type: string
Default: It's beaut!
Description: What do you think of this talk?
 I'd like people's opinion of what they think of this talk.  Is it
 informative?  Is there any point to it?  and so on.
 .
 E-mail can be sent to mpalmer@debian.org after the talk, or see me
 when I'm getting my butt kicked at pool.
```

## Debconf for the maintainer

Access to debconf is quite simple. It is, at it’s core, a text-based pipe between a frontend and your script, and a backend and your script. The two most commonly used languages to write config scripts are Perl and Bourne Shell. We’ll describe both interfaces, since they’re very similar mechanically. I’ll always list the Bourne shell first, for no good reason.

## Initialization

We must initialise the database before we do anything with it.

```
. /usr/share/debconf/confmodule
db_version 2.0
db_capb backup

use Debconf::Client::ConfModule ’:all’;
use Debconf::Log ’:all’;
version(’2.0’);
capb(’backup’);
```

## Asking Questions

Since your questions have already been defined, asking one is as simple as giving your question ID to Debconf and saying “well, ask it already!”.

```
db_input high foo/bar/myquestion || true
db_go || true
```

```
input("high", "foo/bar/myquestion");
go();
```

This will ask the question if the user has requested to see questions of this priority. Else it will skip it and save the default value to the debconf database.

## Giving your question a title

Just put the following code before you actually ask the question.

```
db_title "Is it good?"
```

```
title("Is it good?");
```

## Getting the Answers

Again, it's just retrieving the answer based on the question ID.

```
db_get foo/bar/myquestion
echo "The answer was $RET"
```

```
my @ret = get("foo/bar/myquestion");
print "The answer was $ret[1]\n";
```

## Putting it together

So, the complete ask/retrieve for a question might look like this:

```
. /usr/share/debconf/confmodule
db_version 2.0
db_capb backup

db_input high foo/bar/myquestion || true
db_title "Is it good?"
db_go || true

db_get foo/bar/myquestion
echo "The answer was $RET"
```

# Going around again

One question remains - what if the user fluffed an answer and wants to go back and re-answer a question? They can always abort the config and start again, but that can be a pain in long config scripts. Instead, we can just use the ‘backup’ capability we set earlier.

A db_go or go( ) call will return a numeric value of 30 if the user selected to go back to the previous question. There is no means of going back multiple questions in one hit. It’s up to your script to work out what to do when you go back.

```
db_go
if [ $RET == 30 ]; then
# Do the going back thing
fi
```

```
my @ret = go();
if ($ret[0] == 30) {
# Do the going back thing
}
```

## A means of going back

The next slide shows the core of a very simple and straightforward backup-supporting state machine, in Perl. The mechanics of this system are quite simple. If the state subroutine returns the name of a state, we move to that state. Else we take the previous state off the stack and run that one again.

Every possible state should be represented by a subroutine of the same name, which returns either the name of the next state to be called, or undef if we're to go back.

```
push(my @STATESTACK, "exit");
# Replace with the name of the first state
my $STATE = "intro";

while ($STATE ne "exit") {
        no strict ’refs’;
        my $NEXTSTATE = &$STATE;
        use strict;

        if ($NEXTSTATE) {
                push(@STATESTACK, $STATE);
        } else {
                $NEXTSTATE = pop(@STATESTACK);
        }
        $STATE=$NEXTSTATE;
}
```

# An example state

```
sub intro
{
        title("PHPWiki Configuration");
        input("low", "phpwiki/notes/introduction");
        my @ret=go();
        if ($ret[0] == 30) {
                return undef;
        } else {
                return "docroot";
        }
}
```

## Another example state

```
sub mailname
{
        title("The system mailname");
        input("critical", "shared/mail/mailname");
        my @ret=go();
        if ($ret[0] == 30) {
                return undef;
        } else {
                @ret = get("shared/mail/mailname");
                if (check_rfc1035($ret[1])) {
                        return "write_aliases";
                } else {
                        return "rfc1035";
                }
        }
}
```

```
sub rfc1035
{
        input("critical", "exim/error/rfc1035");
        go();
        return undef;
}

sub check_rfc1035
{
        my $rfc1035_label_re = '[0-9A-Za-z]([-0-9A-Za-z]*[0-9A-Za-z])?';
        my $rfc1035_domain_re = "$rfc1035_label_re(\\.$rfc1035_label_re)

        my $domain = shift;

        return $domain =~ m/^$rfc1035_domain_re$/;
}
```

## A Packaging Example

I'm going to cheat a little and use a package of my own as the example. There are lots of examples of debconf usage - just look at any package which depends on Debconf.

## Debconf for Admins

The default debconf setup stores all questions and answers in a straightforward flat-file database (`/var/cache/debconf/config.dat` and `templates.dat`, for the curious). But debconf can do all sorts of other funky things, like:

## Debconf for Admins

The default debconf setup stores all questions and answers in a straightforward flat-file database (`/var/cache/debconf/config.dat` and `templates.dat`, for the curious). But debconf can do all sorts of other funky things, like:

- Store configuration and/or questions in other formats (such as a directory tree, flat directory, or LDAP directory).

## Debconf for Admins

The default debconf setup stores all questions and answers in a straightforward flat-file database (`/var/cache/debconf/config.dat` and `templates.dat`, for the curious). But debconf can do all sorts of other funky things, like:

- Store configuration and/or questions in other formats (such as a directory tree, flat directory, or LDAP directory).
- “Stack” multiple databases on top of each other, so data items across multiple databases can be accessed as though they were all in one DB.

- Mark databases read-only, so config changes won’t be saved.

- Mark databases read-only, so config changes won’t be saved.
- Store questions and their answers in entirely different places all together.

## Hack me Harder, baby

Adding new frontends and backends is really as simple as writing a new perl module and sticking it in the right place. Frontends (that is, user interfaces) can be added into `/usr/share/perl5/Debconf/FrontEnd`, and backends (data stores) can be added to `/usr/share/perl5/Debconf/DbDriver`. I recommend you submit your creation to Joey Hess (Debconf maintainer) for inclusion in the main package. One currently lacking feature is the ability to store data in an SQL database.

The programmers' documentation is pretty decent - but only exists in the source package. Check it out from a mirror near you.

## References

- The Debconf Programmer's Tutorial (in `debconf-doc`)
- Configuration Management (The Debconf Policy)
- `debconf.conf`(5)