NAME

dconf − A configuration system

DESCRIPTION

dconf is a simple key/value storage system that is heavily optimised for reading. This makes it an ideal system for storing user preferences (which are read 1000s of times for each time the user changes one). It was created with this usecase in mind.

All preferences are stored in a single large binary file. Layering of preferences is possible using multiple files (ie: for site defaults). Lock−down is also supported. The binary file for the defaults can optionally be compiled from a set of plain text keyfiles.

dconf has a partial client/server architecture. It uses D−Bus. The server is only involved in writes (and is not activated in the user session until the user modifies a preference). The service is stateless and can exit freely at any time (and is therefore robust against crashes). The list of paths that each process is watching is stored within the D−Bus daemon itself (as D−Bus signal match rules).

Reads are performed by direct access (via mmap) to the on−disk database which is essentially a hashtable. For this reason, dconf reads typically involve zero system calls and are comparable to a hashtable lookup in terms of speed. Practically speaking, in simple non−layered setups, dconf is less than 10 times slower than GHashTable.

Writes are assumed only to happen in response to explicit user interaction (like clicking on a checkbox in a preferences dialog) and are therefore not optimised at all. On some file systems, dconf−service will call fsync() for every write, which can introduce a latency of up to 100ms. This latency is hidden by the client libraries through a clever "fast" mechanism that records the outstanding changes locally (so they can be read back immediately) until the service signals that a write has completed.

The binary database format that dconf uses by default is not suitable for use on NFS, where mmap does not work well. To handle this common use case, dconf can be configured to place its binary database in **XDG_RUNTIME_DIR** (which is guaranteed to be local, but non−persistent) and synchronize it with a plain text keyfile in the users home directory.

PROFILES

A profile is a list of configuration databases that dconf consults to find the value for a key. The user's personal database always takes the highest priority, followed by the system databases in the order prescribed by the profile.

On startup, dconf consults the **DCONF_PROFILE** environment variable. If set, dconf will attempt to open the named profile, aborting if that fails. If the environment variable is not set, it will attempt to open the profile named "user" and if that fails, it will fall back to an internal hard−wired configuration. dconf stores its profiles in text files. **DCONF_PROFILE** can specify a relative path to a file in /etc/dconf/profile/, or an absolute path (such as in a user's home directory). The profile name can only use alphanumeric characters or '_'.

A profile file might look like the following:

```
user−db:user
system−db:local
system−db:site
```

Each line in a profile specifies one dconf database. The first line indicates the database used to write changes, and the remaining lines indicate read−only databases. (The first line should specify a user−db or service−db, so that users can actually make configuration changes.)

A "user−db" line specifies a user database. These databases are found in **$XDG_CONFIG_HOME**/dconf/. The name of the file to open in that directory is exactly as it is written in the profile. This file is expected to be in the binary dconf database format. Note that **XDG_CONFIG_HOME** cannot be set/modified per terminal or session, because then the writer and reader would be working on different DBs (the writer is started by DBus and cannot see that variable).

A "service−db" line instructs dconf to place the binary database file for the user database in **XDG_RUNTIME_DIR**. Since this location is not persistent, the rest of the line instructs dconf how to store the database persistently. A typical line is service−db:keyfile/user, which tells dconf to synchronize the binary database with a plain text keyfile in **$XDG_CONFIG_HOME**/dconf/user.txt. The synchronization is bi−directional.

A "system−db" line specifies a system database. These databases are found in /etc/dconf/db/. Again, the name of the file to open in that directory is exactly as it is written in the profile and the file is expected to be in the dconf database format.

If the **DCONF_PROFILE** environment variable is unset and the "user" profile can not be opened, then the effect is as if the profile was specified by this file:

```
user−db:user
```

That is, the user's personal database is consulted and there are no system settings.

KEY FILES

To facilitate system configuration with a text editor, dconf can populate databases from plain text keyfiles. For any given system database, keyfiles can be placed into the /etc/dconf/db/*database*.d/ directory. The keyfiles contain groups of settings as follows:

```
# Some useful default settings for our site

[system/proxy/http]
host='172.16.0.1'
enabled=true

[org/gnome/desktop/background]
picture−uri='file:///usr/local/rupert−corp/company−wallpaper.jpeg'
```

After changing keyfiles, the database needs to be updated with the **dconf**(1) tool.

LOCKS

System databases can contain 'locks' for keys. If a lock for a particular key or subpath is installed into a database then no database listed above that one in the profile will be able to modify any of the affected settings. This can be used to enforce mandatory settings.

To add locks to a database, place text files in the /etc/dconf/db/*database*.d/locks directory, where *database* is the name of a system database, as specified in the profile. The files contain list of keys to lock, on per line. Lines starting with a # are ignored. Here is an example:

```
# prevent changes to the company wallpaper
/org/gnome/desktop/background/picture−uri
```

After changing locks, the database needs to be updated with the **dconf**(1) tool.

PORTABILITY

dconf mostly targets Free Software operating systems. It will theoretically run on Mac OS but there isn't much point to that (since Mac OS applications want to store preferences in plist files). It is not possible to use dconf on Windows because of the inability to rename over a file that's still in use (which is what the dconf−service does on every write).

API STABILITY

The dconf API is not particularly friendly, and is not guaranteed to be stable. Because of this and the lack of portability, you almost certainly want to use some sort of wrapper API around it. The wrapper API used by GTK+ and GNOME applications is **GSettings**[1], which is included as part of GLib. GSettings has

backends for Windows (using the registry) and Mac OS (using property lists) as well as its dconf backend and is the proper API to use for graphical applications.

SEE ALSO

dconf-service(1), **dconf-editor**(1), **dconf**(1), **GSettings**[1]

NOTES

1. GSettings
   http://developer.gnome.org/gio/stable/GSettings.html