GNU dmd Manual

For use with dmd 0.2
Last updated 24 June 2014

Wolfgang Jährling
Ludovic Courtès

Table of Contents

1 Introduction

This manual documents the *dmd* service manager. It is used to start and stop system services (typically daemons) in a reliable fashion. For instance it will dynamically determine and start any other services that our desired service depends upon. As another example, dmd might detect conflicts between services. In this situation it would simply prevent the conflicting services from running concurrently.

dmd is the *init system* of the GNU operating system—it is the first user process that gets started, typically with PID 1, and runs as `root`. Normally the purpose of init systems is to manage all system-wide services, but dmd can also be a useful tool assisting unprivileged users in the management of their own daemons.

Unfortunately all flexible software requires some time to master and dmd is no different. But don't worry: this manual should allow you to get started quickly. Its first chapter is designed as a practical introduction to dmd and should be all you need for everyday use (see Chapter 2 [Jump Start], page 2). In chapter two we will describe the `deco` and `dmd` programs, and their relationship, in more detail (Chapter 3 [deco and dmd], page 5). Subsequent chapters provide a full reference manual and plenty of examples, covering all of dmd's capabilities. Finally, the last chapter provides information for those souls brave enough to hack dmd itself.

The name dmd stands for *Daemon Managing Daemons* (or *Daemons-Managing Daemon*?).

This program is written in *Guile*, an implementation of the Scheme programming language, using the *GOOPS* extension for object-orientation. Guile is also dmd's configuration language (see *GNU Guile Reference Manual*). We have tried to make dmd's basic features as accessible as possible—you should be able to use these even if you do not know how to program in Scheme. A basic grasp of Guile and GOOPS is required only if you wish to make use of dmd's more advanced features.

2 Jump Start

This chapter gives a short overview of dmd. It is enough if you just need the basic features of it. As it is not assumed that readers are familiar with all the involved issues, a very experienced user might be annoyed by the often very detailed descriptions in this introduction. Those users are encouraged to just skip to the reference section.

Note that all the full file names in the following text are based on the assumption that you have installed dmd with an empty prefix. If your dmd installation for example resides in `/usr/local` instead, add this directory name in front of the absolute file names mentioned below.

When dmd gets started, it reads and evaluates a configuration file. When it is started with superuser priviledges, it tries to use `/etc/dmdconf.scm`, when started as normal user, it looks for a file called `.dmdconf.scm` in the user's home directory. With the option `--config` (or, for short, `-c`), you can specify where to look instead. So if you want to start dmd with an alternative file, use one of the following commands:

```
dmd --config=/etc/dmdconf.scm.old
dmd -c /etc/dmdconf.scm.old
```

As its name suggests, dmd is just a daemon that (usually) runs in the background, so you will not interact with it directly. After it is started, dmd will listen on a socket special file, usually `/var/run/dmd/socket`, for further commands. You use the tool *deco* to send these commands to dmd. Usage of deco is simple and straightforward: To start a service called `apache`, you use:

```
deco start apache
```

When you do this, all its dependencies will get resolved. For example, a webserver is quite likely to depend on working networking, thus it will depend on a service called `networking`. So if you want to start `apache`, and `networking` is not yet running, it will automatically be started as well. The current status of all the services defined in the configuration file can be queried like this:

```
deco status dmd
```

Or, to get additional details about each service, run:

```
deco detailed-status dmd
```

In this example, this would show the `networking` and `apache` services as started. If you just want to know the status of the `apache` service, run:

```
deco status apache
```

You can stop a service and all the services that depend on it will be stopped. Using the example above, if you stop `networking`, the service `apache` will be stopped as well—which makes perfect sense, as it cannot work without the network being up. To actually stop a service, you use the following, probably not very surprising, command:

```
deco stop networking
```

There are two more actions you can perform on every service: The actions `enable` and `disable` are used to prevent and allow starting of the particular service. If a service is intended to be restarted whenever it terminates (how this can be done will not be covered in this introduction), but it is respawning too often in a short period of time (by default

5 times in 5 seconds), it will automatically be disabled. After you have fixed the problem that caused it from being respawned too fast, you can start it again with the commands:

```
deco enable foo
deco start foo
```

But there is far more you can do than just that. Services can not only simply depend on other services, they can also depend on *virtual* services. A virtual service is a service that is provided by one or more service additionally. For instance, a service called `exim` might provide the virtual service `mailer-daemon`. That could as well be provided by a service called `smail`, as both are mailer-daemons. If a service needs any mailer-daemon, no matter which one, it can just depend on `mailer-daemon`, and one of those who provide it gets started (if none is running yet) when resolving dependencies. The nice thing is that, if trying to start one of them fails, dmd will go on and try to start the next one, so you can also use virtual services for specifying *fallbacks*.

Additionally to all that, you can perform service-specific actions. Coming back to our original example, `apache` is able to reload its modules, therefore the action `reload-modules` might be available:

```
deco reload-modules apache
```

The service-specific actions can only be used when the service is started, i.e. the only thing you can do to a stopped service is starting it. An exception exists, see below. (If you may at some point find this too restrictive because you want to use variants of the same service which are started in different ways, consider using different services for those variants instead, which all provide the same virtual service and thus conflict with each other, if this is desired. That's one of the reasons why virtual services exist, after all.)

There are two actions which are special, because even if services can implement them on their own, a default implementation is provided by dmd (another reason why they are special is that the default implementations can be called even when the service is not running; this inconsistency is just to make it more intuitive to get information about the status of a service, see below).

These actions are `restart` and `status`. The default implementation of `restart` calls `stop` and `start` on the affected service in order, the `status` action displays some general information about the service, like what it provides, what it depends on and with which other services it conflicts (because they provide a virtual service that is also provided by that particular service).

Another special action is `list-actions`, which displays a list of the additional actions a service provides; obviously, it can also be called when the service is not running. Services cannot provide their own implementation of `list-actions`.

A special service is `dmd`, which is used for controlling dmd itself. It implements various actions. For example, the `status` action displays which services are started and which ones are stopped, whereas `detailed-status` has the effect of applying the default implementation of `status` to all services one after another. The `load` action is unusual insofar as it shows a feature that is actually available to all services, but which we have not seen yet: It takes an additional argument. You can use `load` to load arbitrary code into dmd at runtime, like this:

```
deco load dmd ~/additional-services.scm
```

This is enough now about the `deco` and `dmd` programs, we will now take a look at how to configure dmd. In the configuration file, we need mainly the definition of services. We can also do various other things there, like starting a few services already.

FIXME: Finish. For now, look at the `examples/` subdirectory.

. . .

Ok, to summarize:

- dmd is a daemon, deco the program that controls it.
- You can start, stop, restart, enable and disable every service, as well as display its status.
- You can perform additional service-specific actions, which you can also list.
- Actions can have arguments.
- You can display the status of a service, even if the service does not provide a specific implementation for this action. The same is true for restarting.
- The `dmd` service is used to control dmd itself.

3 deco and dmd

The daemon that runs in the background and is responsible for controlling the services is *dmd*, while the user interface tool is called *deco*, the *DaEmon COntroller*[1]. To perform an action, like stopping a service or calling an action of a service, you use the deco program. It will communicate with dmd over a Unix Domain Socket.

Thus, you start dmd once, and then always use deco whenever you want to do something service-related. Since deco passes its current working directory to dmd, you can pass relative file names without trouble. Both dmd and deco understand the standard arguments `--help`, `--version` and `--usage`.

3.1 Invoking dmd

The `dmd` program has the following synopsis:

```
dmd [option...]
```

It accepts the following options:

'`-c file`'
'`--config=file`'
: Read and evaluate *file* as the configuration script on startup.

  file is evaluated in the context of a fresh module where bindings from the `(dmd service)` module and Guile's `(oop goops)` are available, in addition to the default set of Guile bindings. In particular, this means that code in *file* may use `register-services`, the `<service>` class, and related tools (see Chapter 4 [Services], page 7).

'`-I`'
'`--insecure`'
: Do not check if the directory where the socket—our communication rendez-vous with `deco`—is located has permissions `700`. If this option is not specified, `dmd` will abort if the permissions are not as expected.

'`-l [file]`'
'`--logfile[=file]`'
: Log output into *file*, or if *file* is not given, `/var/log/dmd.log` when running as superuser, *~/.dmd.log* otherwise.

'`--pid[=file]`'
: When dmd is ready to accept connections, write its PID to *file* or to the standard output if *file* is omitted.

'`-p [file]`'
'`--persistency[=file]`'
'`-s file`'
'`--socket=file`'
: Receive further commands on the socket special file *file*. If this option is not specified, *`localstatedir`*`/run/dmd/socket` is taken.

[1] Some people might argue that it actually is short for "decoration", indicating that it is useless. :-)

If `-` is specified as file name, commands will be read from standard input, one per line, as would be passed on a `deco` command line (see Section 3.2 [Invoking deco], page 6).

'`--quiet`' Synonym for `--silent`.

3.2 Invoking deco

The `deco` command is a generic client program to control a running instance of `dmd` (see Section 3.1 [Invoking dmd], page 5). It has the following synopsis:

```
deco [option...] action service [arg...]
```

It causes the *action* of the *service* to be invoked. For each action, you should pass the appropriate *arg*s. Actions that are available for every service are `start`, `stop`, `restart`, `status`, `enable`, `disable`, and `doc`.

If you pass a file name as an *arg*, it will be passed as-is to dmd, thus if it is not an absolute name, it is local to the current working directory of dmd, not to deco.

The `deco` command understands the following option:

'`-s file`'
'`--socket=file`'
: Send commands to the socket special file *file*. If this option is not specified, *localstatedir*`/run/dmd/socket` is taken.

3.3 Invoking reboot

The `reboot` command is a convenience client program to instruct dmd (when used as an init system) to stop all running services and reboot the system. It has the following synopsis:

```
reboot [option...]
```

It is equivalent to running `deco stop dmd`. The `reboot` command understands the following option:

'`-s file`'
'`--socket=file`'
: Send commands to the socket special file *file*. If this option is not specified, *localstatedir*`/run/dmd/socket` is taken.

3.4 Invoking halt

The `halt` command is a convenience client program to instruct dmd (when used as an init system) to stop all running services and turn off the system. It has the following synopsis:

```
halt [option...]
```

It is equivalent to running `deco power-off dmd`. As usual, the `halt` command understands the following option:

'`-s file`'
'`--socket=file`'
: Send commands to the socket special file *file*. If this option is not specified, *localstatedir*`/run/dmd/socket` is taken.

4 Services

The *service* is obviously a very important concept of dmd. On the Guile level, a service is represented as an instance of `<service>`, a GOOPS class (see Section "GOOPS" in *GNU Guile Reference Manual*). When creating an instance of it, you can specify the initial values of its slots, and you actually must do this for some of the slots.

The `<service>` class and its associated procedures and methods are defined in the `(dmd service)` module.

4.1 Slots of services

A service has the following slots, all of which can be initialized with a keyword (i.e. `#:provides`, used when creating the object) of the same name, except where stated otherwise. You should not access them directly with `slot-ref` or `slot-set!` usually, use the methods of the service class Section 4.2 [Methods of services], page 8 instead.

- `provides` is a list of symbols that are provided by the service. A symbol can only be provided by one service running at a time, i.e. if two services provide the same symbol, only one of them can run, starting the other one will fail. Therefore, these symbols are mainly used to denote conflicting services. The first symbol in the list is the canonical name for the service, thus it must be unique. This slot has no default value and must therefore be initialized.
- `requires` is, like `provides`, a list of symbols that specify services. In this case, they name what this service depends on, i.e. before the service can be started, services that provide those symbols must be started. If a required symbol is provided by several services, one will be started. By default, this slot contains the empty list.
- `running` is a hook that can be used by each service in its own way. The default value is `#f`, which indicates that the service is not running. When an attempt is made to start the service, it will be set to the return value of the procedure in the `start` slot. It will also be passed as an argument to the procedure in the `stop` slot. This slot can not be initialized with a keyword.
- `respawn?` specifies whether the service should be respawned by dmd. If this slot has the value `#t`, then assume the `running` slot specifies a child process PID and restart the service if that process terminates. Otherwise this slot is `#f`, which is the default. See also the `last-respawns` slot.
- `start` contains the "constructor" for the service, which will be called to start the service. (Therefore, it is not a constructor in the sense that it initializes the slots of a `<service>` object.) This must be a procedure that accepts any amount of arguments, which will be the additional arguments supplied by the user. If the starting attempt failed, it must return `#f`. The value will be stored in the `running` slot. The default value is a procedure that returns `#t` and performs no further actions, therefore it is desirable to specify a different one usually.
- `stop` is, similar to `start`, a slot containing a procedure. But in this case, it gets the current value of the `running` slot as first argument and the user-supplied arguments as further arguments; it gets called to stop the service. Its return value will again be stored in the `running` slot, so that it should return `#f` if it is now possible again to

start the service at a later point. The default value is a procedure that returns `#f` and performs no further actions.

- `actions` specifies the additional actions that can be performed on a service when it is running. A typical example for this is the `restart` action. The macro `make-actions` Section 4.3 [Service Convenience], page 9 is provided to abstract the actual data representation format for this slot. (It actually is a hash currently.)
- `enabled?` cannot be initialized with a keyword, and contains `#t` by default. When the value becomes `#f` at some point, this will prevent the service from getting started. A service can be enabled and disabled with the methods `enable` and `disable`, respectively Section 4.2 [Methods of services], page 8.
- `last-respawns` cannot be initialized with a keyword and is only ever used when the `respawn?` slot contains `#t`; it is a circular list with `(car respawn-limit)` elements, where each element contains the time when it was restarted, initially all 0, later a time in seconds since the Epoch. The first element is the one that contains the oldest one, the last one the newest.
- `stop-delay?` being false causes the `stop` slot to be unused; instead, stopping the service will just cause the `waiting-for-termination?` slot be set to `#t`.
- `waiting-for-termination?` cannot be initialized with a keyword and should not be used by others, it is only used internally for respawnable services when the `stop-delay?` slot contains a true value. `waiting-for-termination?` contains `#t` if the service is still running, but the user requested that it be stopped, in which case if the service terminates the next time, the respawn handler will not start it again.

  otherwise `#f`.

4.2 Methods of services

`start` (*obj* <*service*>) [method]

Start the service *obj*, including all the services it depends on. It tries quite hard to do this: When a service that provides a required symbol can not be started, it will look for another service that also provides this symbol, until starting one such service succeeds. There is some room for theoretical improvement here, of course, but in pratice the current strategy already works very well. This method returns the new value of the `running` slot Section 4.1 [Slots of services], page 7, which is `#f` if the service could not be started.

`stop` (*obj* <*service*>) [method]

This will stop the service *obj*, trying to stop services that depend in it first, so they can be shutdown cleanly. If this will fail, it will continue anyway. Stopping of services should usually succeed, though. Otherwise, the behaviour is very similar to the `start` method. The return value is also the new `running` value, thus `#f` if the service was stopped.

`action` (*obj* <*service*>) *the-action* . *args* [method]

Calls the action *the-action* (a symbol) of the service *obj*, with the specified *args*, which have a meaning depending on the particular action.

`conflicts-with` (*obj* <*service*>) [method]
: Returns a list of the canonical names of services that conflict with the service *obj*.

`canonical-name` (*obj* <*service*>) [method]
: Returns the canonical name of *obj*, which is the first element of the `provides` list.

`provided-by` (*obj* <*service*>) [method]
: Returns which symbols are provided by *obj*.

`required-by` (*obj* <*service*>) [method]
: Returns which symbols are required by *obj*.

`running?` (*obj* <*service*>) [method]
: Returns whether the service *obj* is running.

`respawn?` (*obj* <*service*>) [method]
: Returns whether the service *obj* should be respawned if it terminates.

`default-display-status` (*obj* <*service*>) [method]
: Display status information about *obj*. This method is called when the user performs the action `status` on *obj*, but there is no specific implementation given for it. It is also called when `detailed-status` is applied on `dmd`.

4.3 Service Convenience

In addition to the facilities listed below, there are also some procedures that provide commonly needed constructors and destructors for services Section 4.4 [Service De- and Constructors], page 10.

`register-services` . *services* [procedure]
: Register all *services*, so that they can be taken into account when trying to resolve dependencies.

`lookup-services` *name* [procedure]
: Return a list of all registered services which provide the symbol *name*.

`make-actions` (*name proc*) ... [macro]
: This macro is used to create a value for the `actions` slot of a service object Section 4.1 [Slots of services], page 7. Each *name* is a symbol and each *proc* the corresponding procedure that will be called to perform the action. A *proc* has one argument, which will be the current value of the `running` slot of the service.

`start` (*obj* <*symbol*>) [method]
: Start a registered service providing *obj*.

`stop` (*obj* <*symbol*>) [method]
: Stop a registered service providing *obj*.

`action` (*obj* <*symbol*>) *the-action* . *args* [method]
: The same as the `action` method of class `<service>`, but uses a service that provides *obj* and is running.

`for-each-service` *proc* [procedure]
Call *proc*, a procedure taking one argument, once for each registered service.

`find-running` *services* [procedure]
Check if any of *services* is running. If this is the case, return its canonical name. If not, return `#f`. Only the first one will be returned; this is because this is mainly intended to be applied on the return value of `lookup-services`.

4.4 Service De- and Constructors

All of the procedures listed below return procedures generated from the supplied arguments. These procedures take one argument in the case of destructors and no arguments in the case of constructors.

`make-system-constructor` *command...* [procedure]
The returned procedure will execute *command* in a shell and return `#t` if execution was successful, otherwise `#f`. For convenience, it takes multiple arguments which will be concatenated first.

`make-system-destructor` *command...* [procedure]
Similar to `make-system-constructor`, but returns `#f` if execution of the *command* was successful, `#t` if not.

`make-forkexec-constructor` *command* [*#:directory* (*default-service-directory*)] [*#:environment-variables* (*default-environment-variables*)] [procedure]
Return a procedure that forks a child process, close all file descriptors except the standard output and standard error descriptors, sets the current directory to *directory*, changes the environment to *environment-variables* (using the `environ` procedure), and executes *command* (a list of strings.) Return the PID of the child process.

`make-kill-destructor` [*signal*] [procedure]
Returns a procedure that sends *signal* to the pid which it takes as argument. This *does* work together with respawning services, because in that case the `stop` method of the `<service>` class sets the `running` slot to `#f` before actually calling the destructor; if it would not do that, killing the process in the destructor would immediately respawn the service.

The `make-forkexec-constructor` procedure builds upon the following procedures.

`exec-command` *command* [*#:directory* (*default-service-directory*)] [*#:environment-variables* (*default-environment-variables*)] [procedure]
`fork+exec-command` *command* [*#:directory* (*default-service-directory*)] [*#:environment-variables* (*default-environment-variables*)] [procedure]
Run *command* as the current process from *directory*, and with *environment-variables* (a list of strings like `"PATH=/bin"`.) File descriptors 1 and 2 are kept as is, whereas file descriptor 0 (standard input) points to `/dev/null`; all other file descriptors are closed prior to yielding control to *command*.

`fork+exec-command` does the same, but in a separate process whose PID it returns.

4.5 Service Examples

FIXME: This needs a lot of work.

You can create a service and then register it this way:

```
(define apache (make <service>
                     #:provides '(apache)
                     #:start (...)
                     #:stop (...)))
(register-services apache)
```

However, as you usually won't need a variable for the service, you can pass it directly to `register-services`. Here is an example that also specifies some more initial values for the slots:

```
(register-services
  (make <service>
        #:provides '(apache-2.0 apache httpd)
        #:requires '()
        #:start (...)
        #:stop (...)
        #:actions (make-actions
                   (reload-modules (...))
                   (restart (...)))))
```

4.6 The `dmd` and `unknown` services

The service `dmd` is special, because it is used to control dmd itself. It provides the following actions (in addition to `enable`, `disable` and `restart` which do not make sense here).

`status` Displays which services are started and which ones are not.

`detailed-status`
Displays detailed information about every registered service.

`load` *`file`* Evaluate the Scheme code in *file* in a fresh module that uses the `(oop goops)` and `(dmd services)` modules—as with the `--config` option of `dmd` (see Section 3.1 [Invoking dmd], page 5).

`unload` *`service-name`*
Attempt to remove the service identified by *service-name*. `dmd` will first stop the service, if necessary, and then remove it from the list of registered services. Any services depending upon *service-name* will be stopped as part of this process.

If *service-name* simply does not exist, output a warning and do nothing. If it exists, but is provided by several services, output a warning and do nothing. This latter case might occur for instance with the fictional service `web-server`, which might be provided by both `apache` and `nginx`. If *service-name* is the special string and `all`, attempt to remove all services except for dmd itself.

`reload` *`file-name`*
Unload all known optional services using unload's `all` option, then load *file-name* using `load` functionality. If file-name does not exist or `load` encounters

an error, you may end up with no defined services. As these can be reloaded at a later stage this is not considered a problem. If the `unload` stage fails, `reload` will not attempt to load *file-name*.

`daemonize`
: Fork and go into the background. This should be called before respawnable services are started, as otherwise we would not get the `SIGCHLD` signals when they terminate.

`enable-persistency`
: When terminating, safe the list of running services in a file.

`disable-persistency`
: Don't safe the list of running services when terminating.

The `unknown` service must be defined by the user and if it exists, is used as a fallback whenever we try to invoke an unknown action of an existing service or use a service that does not exist. This is useful only in few cases, but enables you to do various sorts of unusual things.

5 Runlevels

RUNLEVELS DO NOT WORK YET! Do not use them! Ignore this section!

A *runlevel* makes it easier to start and stop groups of services, to bring the system into a certain state. An object of class `<runlevel>` is an abstract runlevel, and has the following methods:

`enter` (*rl* `<`*runlevel*`>`) *services* [method]

This will be called when the runlevel should be entered. *services* is the list of the currently running services.

6 Misc Facilities

This is a list of facilities which are available to code running inside of dmd and is considered generally useful, but is not directly related to one of the other topic covered in this manual.

6.1 Errors

`assert` *expr* [macro]
: If *expr* yields `#f`, display an appropriate error message and throw an `assertion-failed` exception.

`caught-error` *key args* [procedure]
: Tell dmd that a *key* error with *args* has occured. This is the simplest way to cause caught error result in uniformly formated warning messages. The current implementation is not very good, though.

`call/cc` *proc* [procedure]
: An alias for `call-with-current-continuation`.

`call/ec` *proc* [procedure]
: A simplistic implementation of the nonstandard, but popular procedure `call-with-escape-continuation`, i.e. a `call/cc` for outgoing continuations only. Note that the variant included in dmd is not aware of `dynamic-wind` at all and does not yet support returning multiple values.

`without-system-error` *expr...* [macro]
: Evaluates the *expr*s, not going further if a system error occurs, but also doing nothing about it.

6.2 Communication

The `(dmd comm)` module provides primitives that allow clients such as `deco` to connect to `dmd` and send it commands to control or change its behavior (see Section 4.1 [Slots of services], page 7).

Currently, clients may only send *commands*, represented by the `<dmd-command>` type. Each command specifies a service it applies to, an action name, a list of strings to be used as arguments, and a working directory. Commands are instantiated with `dmd-command`:

`dmd-command` *action service* [*#:arguments* '()] [*#:directory* (*getcwd*)] [procedure]
: Return a new command (a `<dmd-command>`) object for *action* on *service*.

Commands may then be written to or read from a communication channel with the following procedures:

`write-command` *command port* [procedure]
: Write *command* to *port*.

`read-command` *port* [procedure]
: Receive a command from *port* and return it.

In practice, communication with `dmd` takes place over a Unix-domain socket, as discussed earlier (see Section 3.1 [Invoking dmd], page 5). Clients may open a connection with the procedure below.

`open-connection` [*file*] [procedure]
: Open a connection to the daemon, using the Unix-domain socket at *file*, and return the socket.

  When *file* is omitted, the default socket is used.

The daemon writes output to be logged or passed to the currently-connected client using `local-output`:

`local-output` *format-string . args* [procedure]
: This procedure should be used for all output operations in dmd. It outputs the *args* according to the *format-string*, then inserts a newline. It writes to whatever is the main output target of dmd, which might be multiple at the same time in future versions.

6.3 Others

`copy-hashq-table` *table new-size* [procedure]
: Create a hash-table with size *new-size*, and insert all values from *table* into it, using `eq?` when inserting. This procedure is mainly used internally, but is a generally useful utillity, so it can by used by everyone.

7 Internals

This chapter contains information about the design and the implementation details of dmd for people who want to hack dmd itself. If you want your work to get included in dmd, please contact me and say what you intend to do so that I can give advice on how to do it and we can avoid duplicating work. My development version is usually a bit ahead of what I release, as I only want to publish code that got some testing.

7.1 Coding standards

About formatting: Use common sense and GNU Emacs (which actually is the same, of course), and you almost can't get the formatting wrong. Formatting should be as in Guile and Guix, basically.

7.2 Design decisions

The general idea of a service manager that uses dependencies, similar to those of a Makefile, came from the developers of the GNU Hurd, but as few people are satisfied with System V Init, many other people had the same idea independently. Nevertheless, dmd was written with the goal of becoming a replacement for System V Init on GNU/Hurd, which was one of the reasons for choosing the extension language of the GNU project, Guile, for implementation (another reason being that it makes it just so much easier).

The runlevel concept (i.e. thinking in *groups* of services) is sometimes useful, but often one also wants to operate on single services. System V Init makes this hard: While you can start and stop a service, `init` will not know about it, and use the runlevel configuration as its source of information, opening the door for inconsistencies (which fortunatly are not a practical problem usually). In dmd, this was avoided by having a central entity that is responsible for starting and stopping the services, which therefore knows which services are actually started (if not completely inproperly used, but that is a requirement which is impossible to avoid anyway). While runlevels are not implemented yet, it is clear that they will sit on top of the service concept, i.e. runlevels will merely be an optional extension that the service concept does not rely on. This also makes changes in the runlevel design easier when it may become necessary.

The consequence of having a daemon running that controls the services is that we need another program as user interface which communicates with the daemon. Fortunatly, this makes the commands necessary for controlling services pretty short and intuitive, and gives the additional bonus of adding some more flexibility. For example, it is easiely possible to grant password-protected control over certain services to unprivileged users, if desired.

An essential aspect of the design of dmd (which was already mentioned above) is that dmd should always know exactly what is happening, i.e. which services are started and stopped. The alternative would have been to not use a daemon, but to save the state on the file system, again opening the door for inconsistencies of all sorts. Also, we would have to use a seperate program for respawning a service (which just starts the services, waits until it terminates and then starts it again). Killing the program that does the respawning (but not the service that is supposed to be respawned) would cause horrible confusion. My understanding of "The Right Thing" is that this conceptionally limited strategy is exactly what we do not want.

The way dependencies work in dmd took a while to mature, as it was not easy to figure out what is appropriate. I decided to not make it too sophisticated by trying to guess what the user might want just to theoretically fulfill the request we are processing. If something goes wrong, it is usually better to tell the user about the problem and let her fix it, taking care to make finding solutions or workarounds for problems (like a misconfigured service) easy. This way, the user is in control of what happens and we can keep the implementation simple. To make a long story short, *we don't try to be too clever*, which is usually a good idea in developing software.

If you wonder why I was giving a "misconfigured service" as an example above, consider the following situation, which actually is a wonderful example for what was said in the previous paragraph: Service X depends on symbol S, which is provided by both A and B. A depends on AA, B depends on BB. AA and BB conflict with each other. The configuration of A contains an error, which will prevent it from starting; no service is running, but we want to start X now. In resolving its dependencies, we first try to start A, which will cause AA to be started. After this is done, the attempt of starting A fails, so we go on to B, but its dependency BB will fail to start because it conflicts with the running service AA. So we fail to provide S, thus X cannot be started. There are several possibilities to deal with this:

- When starting A fails, terminate those services which have been started in order to fulfill its dependencies (directly and indirectly). In case AA was running already, we would not want to terminate it. Well, maybe we would, to avoid the conflict with BB. But even if we would find out somehow that we need to terminate AA to eventually start X, is the user aware of this and wants this to happen (assuming AA was running already)? Probably not, she very likely has assumed that starting A succeeds and thus terminating AA is not necessary. Remember, unrelated (running) services might depend in AA. Even if we ignore this issue, this strategy is not only complicated, but also far from being perfect: Let's assume starting A succeeds, but X also depends on a service Z, which requires BB. In that case, we would need to detect in the first place that we should not even try to start A, but directly satisfy X's dependency on S with B.
- We could do it like stated above, but stop AA only if we know we won't need it anymore (for resolving further dependencies), and start it only when it does not conflict with anything that needs to get started. But should we stop it if it conflicts with something that *might* get started? (We do not always know for sure what we will start, as starting a service might fail and we want to fall back to a service that also provides the particular required symbol in that case.) I think that either decision will be bad in one case or another, even if this solution is already horribly complicated.
- When we are at it, we could just calculate a desired end-position, and try to get there by starting (and stopping!) services, recalculating what needs to be done whenever starting a service fails, also marking that particular service as unstartable, except if it fails to start because a dependency could not be resolved (or maybe even then?). This is even more complicated. Instead of implementing this and thereby producing code that (a) nobody understands, (b) certainly has a lot of bugs, (c) will be unmaintainable and (d) causes users to panic because they won't understand what will happen, I decided to do the following instead:
- Just report the error, and let the user fix it (in this case, fix the configuration of A) or

work around it (in this case, disable A so that we won't start AA but directly go on to starting B).

I hope you can agree that the latter solution after all is the best one, because we can be sure to not do something that the user does not want us to do. Software should not run amok. This explanation was very long, but I think it was necessary to justify why dmd uses a very primitive algorithm to resolve dependencies, despite the fact that it could theoretically be a bit more clever in certain situations.

One might argue that it is possible to ask the user if the planned actions are ok with her, and if the plan changes ask again, but especially given that services are supposed to usually work, I see few reasons to make the source code of dmd more complicated than necessary. If you volunteer to write *and* maintain a more clever strategy (and volunteer to explain it to everyone who wants to understand it), you are welcome to do so, of course. . .

7.3 Service Internals

7.4 Runlevel evolution

This section describes how the runlevel concept evolved over time. This is basically a collection of mistakes, but is provided here for your information, and possibly for your amusement, but I'm not sure if all this weird dependency stuff is really that funny.

7.4.1 Runlevel assumptions

A runlevel is a system state, i.e. it consists of the information about which services are supposed to be available and which not. This vague definition implies that several different runlevel styles can be implemented in a service manager.

For example, you can do it like System V Init, specifying which services should be started when we enter a runlevel and which ones should be stopped when leaving it. But one could also specify for every service in which runlevels it should be running.

In dmd, we do not want to limit ourselfes to a single runlevel style. We allow for all possible strategies to be implemented, providing the most useful ones as defaults. We also want to make it possible to combine the different styles arbitrariely.

Therefore, when entering a runlevel, we call a user-defined piece of code, passing it the list of currently active services and expecting as the result a list of service symbols which tell us which services we want to have running. This interface makes it very easy to implement runlevel styles, but makes it not-so-easy for the runlevel implementation itself, because we have to get from the current state into a desired state, which might be more or less vague (since it is not required to be a list of canonical names). Obviously service conflicts and already running services need to be taken into account when deciding which services should be used to provide the various symbols.

Also, the runlevel implementation should be implemented completely on top of the service concept, i.e. the service part should not depend on the idea of runlevels or care about them at all. Otherwise understanding the service part (which is the most essential aspect of dmd) would become harder than necessary.

7.4.2 Runlevels, part one

I came up with the following method (here in Pseudo-Scheme), which is possibly slightly buggy, but should give you the idea:

```
;; Beginning with the canonical names in CURRENT-SERVICES, start and
;; stop services until getting into a state where everything requested
;; in TARGET-SERVICES (which does not only consist of canonical names)
;; is provided, and the things they depends on, but no more.
(define (switch-runlevel current-services target-services)
  (let ((target-services-backup target-services)
(unstartable '()))
    (let retry ()
      (repeat-until-none-of-these-changes-annythig
       ;; Replace all of them with canonical names which provide them.
       (canonicalize-names! target-services unstartable current-services)
       ;; Add what we need additionally.
       (add-dependencies! target-services unstartable current-services))
      (remove-redundancy! target-services)
      (stop-all-unneeded target-services)
      (catch 'service-could-not-be-started
     (lambda ()
       ;; Iterate over the list, starting only those which
       ;; have all dependencies already resolved, so nothing
       ;; we don't want will be started.  Repeat until done.
       (carefully-start target-services))
     (lambda (key service)
       (set! unstartable (cons service unstartable))
       (set! target-services backup-target-services)
       (set! current-services (compute-current-services))
       (retry))))))
```

This indeed looks like a nice way to get what we want. However, the details of this are not as easy as it looks like. When replacing virtual services with canonical names, we have to be very careful. Consider the following situation:

The virtual service X is provided by both A and B, while Y is provided only by B. We want to start C (which depends on X) and D (which depends on Y). Obviously we should use B to fulfill the dependency of C and D on X and Y, respectively. But when we see that we need something that provides X, we are likely to do the wrong thing: Select A. Thus, we need to clean this up later. I wanted to do this as follows:

While substituting virtual services with canonical names, we also safe which one we selected to fulfill what, like this:

```
((A . (X))
 (B . (Y)))
```

Later we look for conflicts, and as A and B conflict, we look which one can be removed (things they provide but are not required by anyone should be ignored, thus we need to create a list like the above). In this case, we can replace A with B as B also provides X (but A does not provide Y, thus the reverse is impossible). If both could be used, we probably

should decide which one to use by looking at further conflicts, which gets pretty hairy. But, in this case, we are lucky and end up with this:

```
((B . (X Y)))
```

This way of finding out which service we should use in case of conflicts sounds pretty sane, but if you think it will work well, you have been fooled, because actually it breaks horribly in the following situation:

Service	Provides
A	W X Y -
B	W X - Z
C	- X Y Z
D	W - - -

If we need all of W, X, Y and Z, then obviously we need to take C and D. But if we have a list like this, we cannot fix it:

```
((A . (W X Y))
 (B . (Z)))
```

Thus, we cannot do it this way.

7.4.3 Runlevels, part two

Let's look again at the table at the end of part two:

Service	Provides
A	W X Y -
B	W X - Z
C	- X Y Z
D	W - - -

If from this table it is so obvious for us what we should do, then it should also be possible to calculate it for a computer, given such a table as input. Ok, we have to take into account conflicts that are not visible in this table, but the general idea is usable. But how do we find which combination works? I found only one way yet: Kind of a brute force attack: Try combinations until we find one that works.

This alone would be too slow. With 20 services we would have 2^20 possible combinations, that is a bit more than a million. Fortunatly, we can optimize this. First I thought we could remove all services from the list that do not provide any symbol we need, but that is obviously a stupid idea, as we might need them for dependencies, in which case we need to take into account their conflicts. But the following method would work:

Very often a symbol that is required will be a canonical name already, i.e. be provided only by a single service. Using our example above, let's suppose we also need the symbol V, which is provided only by D. The first step we do is to look which (required) symbols are provided only by a single service, as we will need this service for sure. In this case, we would need D. But by using it, we would also get the other symbols it provides, W in this case. This means that we don't need to bother looking at other services that provide W, as we cannot use them because they conflict with a service that we definitely need. In this case, we can remove A and B from the list this way. Note that we can remove them entirely, as all their conflicts become irrelevant to us now. In this simple case we would not even have to do much else, C is the only remaining service.

After this first step, there remain the symbols that are provided by two or more services. In every combination we try, exactly one of them must be used (and somehow we should take into account which services are running already). This also reduces the amount of possible combinations a lot. So what remains after that are the services we might need for fulfilling dependencies. For them, we could try all combinations (2^n), making sure that we always try subsets before any of their supersets to avoid starting unneeded services. We should take into account which services are already running as well.

The remaining question is, what to do if starting a service fails. A simple solution would be to recursively remove all services that depend on it directly or indirectly. That might cause undesired side-effects, if a service was running but it had to be stopped because one of the services that provides something it depends on gets exchanged for another service that provides the same symbol, but fails to start. The fact that we would have to stop the (first) service is a problem on its own, though.

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept Index

Procedure and Macro Index

Variable Index

Type Index