SLB User Manual

version 1.0, 26 April 2011

Sergey Poznyakoff.

Short Contents

Table of Contents

1 Introduction

It is a common practice to set up several servers for handling the same task. In such configurations, it is important to distribute the load equally among the computers forming the set. While the term *load* may refer in this context to various things (e.g. CPU load, network traffic, etc.), the main principle of load balancing remains the same: determine which servers are less loaded and transfer to them part of the work from the most loaded ones.

SLB (or *Simple Load Balancer*) is a tool designed for handling this task. It monitors a set of servers and uses SNMP protocol to periodically collect a set of parameters from each of them. Once obtained, these parameters are used as arguments to a *load estimation function*, which computes, for each server, a floating point value representing its *relative load*. SLB then sorts servers in the order of increasing relative load.

SLB does not attempt to actually correct load distribution, as this task depends heavily on the kind of work the servers are performing and it is difficult, if not impossible, to provide a generalized solution for that. Instead, SLB relies on an external program which is supposed to redistribute the load, based on the data obtained from SLB. Therefore, after obtaining the sorted list of servers, SLB filters it through a user-defined format template, and sends the result to a file, a named pipe or another program.

All parameters used by SLB, including load estimation function, are supplied to it in a *configuration file*, which makes the package extremely flexible.

When speaking about the package as a whole, we spell its name as SLB. When speaking about the *program* which constitutes the principal part of the package, we refer to it as `slb`.

2 Overview of this Manual

This book consists of the three main parts. The first part is a tutorial, which provides a gentle (as far as possible) introduction for those who are new to `slb`. Although it is meant to be self-contained, the reader is required to have some basic notions about SNMP and UNIX system administration. The tutorial should help the reader to familiarize himself with the package and to start using it. It does not, however, cover some of the less frequently used features of `slb`.

The two chapters that follow complement the tutorial. The *Invocation* summarizes the command line syntax and options available to `slb`. The *Configuration* contains a detailed reference to the configuration file syntax and statements. These two chapters provide all the information necessary to understand and configure `slb`, for both beginners and for users familiar with the package.

3 Tutorial

This chapter guides you through the most important features of `slb` and explains its configuration. Start here if you have never used `slb` before.

It omits some complicated details and rarely used features, which will be discussed further, in Chapter 5 [Configuration], page 21.

3.1 Option Basics

Options start with a dash. Most of them have two forms, called short and long forms. Both forms are absolutely identical in function; they are interchangeable.

The *short* form is a traditional form for UNIX utilities. In this form, the option consists of a single dash, followed by a single letter, e.g. '`-c`'.

Short options which require arguments take their arguments immediately following the option letter, optionally separated by white space. For example, you might write '`-o name`', or '`-oname`'. Here, '`-o`' is the option, and '`name`' is its argument.

Short options' letters may be clumped together, but you are not required to do this. When short options are clumped as a set, use one (single) dash for them all, e.g. '`-ne`' is equivalent to '`-n -e`'. However, only options that do not take arguments may be clustered this way. If an option takes an argument, it can only be the last option in such a cluster, otherwise it would be impossible to specify the argument for it. Anyway, it is much more readable to specify such options separated.

The *long* options consist of two dashes, followed by a multi-letter option name, which is usually selected to be a mnemonics for the operation it requests. Long option names can be abbreviated, provided that such an abbreviation is unique among the options understood by the program.

Arguments to long options follow the option name being separated from it either by an equal sign, or by any amount of white space characters. For example, the option '`--eval`' with the argument '`test`' can be written either as '`--eval=test`' or as '`--eval test`'.

3.2 Configuration Basics

The *configuration file* defines most parameters needed for the normal operation of `slb`. The program will not start if its configuration file does not exist, cannot be read, or contains some errors.

The configuration file is located in your system configuration directory (normally '`/etc`') and is named '`slb.conf`'. You can place it elsewhere as well, but in this case you will need to explicitly inform `slb` about its actual location, using the '`--config-file`' ('`-c`') command line option:

```
slb --config-file ./new.conf
```

Before actually starting the program, it is wise to check the configuration file for errors. To do so, use the '`--lint`' ('`-t`') command line option:

```
slb --lint
```

When started with this option, `slb` parses the configuration, reports any errors on the standard error and exits. If parsing succeeds, it exits with code 0. Otherwise, if any errors have been found, it exits with code 78 (*configuration error*).

The '`--lint`' option can, of course, be used together with '`--config-file`', e.g.:

```
slb --lint --config-file ./new.conf
```

If you are unsure about the correct configuration syntax, you can obtain a concise summary any time, by running:

```
slb --config-help
```

The summary is printed to the standard output and includes all configuration statements with short descriptions of their purpose and arguments.

In this section we will provide a quick start introduction to the `slb` configuration. For a more detailed and formal discussion, refer to Chapter 5 [Configuration], page 21.

The configuration file consists of *statements*. There are two kinds of statements, called simple and block statements. A *simple statement* consists of a keyword and value, or values, separated by any amount of whitespace and terminated with a semicolon, for example:

```
wakeup 15;
```

A *block statement* is used for logical grouping of other statements. It consists of a keyword, optionally followed by a value, and a set of other statements, enclosed in a pair of curly brackets. For example:

```
syslog {
  facility local1;
  tag slb;
}
```

A semicolon may follow the closing '`}`', although this is not required.

Note that whitespace (i.e. space characters, tabs and newlines) has no special syntactical meaning, except that it serves to separate otherwise adjacent tokens. For example, the following form of the '`syslog`' statement is entirely equivalent to the one shown above:

```
syslog { facility local1;   tag  slb; }
```

Several types of *comments* are supported. A *single-line* comment starts with '`#`' or '`//`' and continues to the end of the line. A *multi-line* or *C-style* comment starts with the two characters '`/*`' (slash, star) and continues until the first occurrence of '`*/`' (star, slash). Whatever comment type are used, they are removed from the configuration prior to parsing it.

After comment removal, the configuration is *preprocessed* using `m4`. This is a highly useful feature, which allows for considerable simplification of configuration files. It is described in Section 3.7 [m4], page 10.

3.3 Daemon Configuration

The most important daemon parameters are the operation mode and wakeup interval. You are not required to explicitly configure them, but you may want to do so, if their default values don't suit you.

There are two *operation modes*: '`standalone`' and '`cron`'. In standalone mode, `slb` detaches itself from the controlling terminal and continues operation in the background, as a *daemon*. It will periodically wake up, poll the servers, compute load table and format it to the output file or pipe. This is the default operation mode.

In *cron* mode, `slb` starts, polls the servers, computes and formats load table and exits when done. This mode is designed for use from crontabs, hence its name. It has some limitation, compared to the standalone mode. Most notably, the `d()` function (derivative) is not available in this mode.

If you wish to explicitly set the operation mode, use the '`standalone`' configuration statement. The statement:

```
standalone yes;
```

configures the standalone mode, whereas the statement:

```
standalone no;
```

configures cron mode. The later can also be requested from the command line, using the '`--cron`' option.

When operating in standalone mode, the *wakeup interval* specifies the amount of time, in seconds, between successive polls. The default value is 5 minutes. To set another value, use the '`wakeup`' statement. For example, the following configures `slb` to do a poll each minute:

```
wakeup 60;
```

3.4 SNMP Configuration

Normally, this does not require additional configuration, unless you want to load some custom MIBs.

To load an additional MIB file, use the '`add-mib`' statement. Its argument is the name of the file to load.

```
add-mib "MY-MIBS.txt";
```

Unless full pathname is specified, the file is searched in the SNMP search path. To modify the search path, use the '`mib-directory`'. Each '`mib-directory`' adds a single directory to the end of the path:

```
mib-directory "/usr/share/slb/mibs";
```

3.5 Server Definition

Server definitions are principal part of any `slb` configurations. They define remote servers for which `slb` is to compute the load table. A server definition is a block statement which begins as:

```
server id {
```

The *id* argument specifies *server ID*, an arbitrary string of characters uniquely identifying this server. This ID will be used by log messages concerning this server. It can also be referred to in the output format definition (see Section 5.7 [output], page 32).

As usual, the definition ends with a closing '`}`'.

Statements inside the '`server`' block supply configuration parameters for this server. Two parameters are mandatory for each server: its IP address and SNMP community:

```
host ip;
community string;
```

The *ip* argument can be either the IP address of the server in dotted-quad notation, or its host name. For example:

```
host 192.168.10.6;
community "public";
```

The most important part of a server definition is its *load estimation function*. It is basically an arithmetic expression of arbitrary complexity, with SNMP variables serving as its terms (see Section 5.4 [expression], page 26). It is defined using the '`expression`' statement.

To begin with, suppose you want to use 1-minute load average as relative load. Let's name it '`la1`'. Then, the expression is very simple and can be defined as:

```
expression "la1";
```

Now, '`la1`' is a variable name, which should be bound to the corresponding SNMP variable. This binding is declared with the '`variable`' statement:

```
variable la1 "UCD-SNMP-MIB::laLoadFloat.1";
```

The '`variable`' statement has two arguments. The first one is the name of a variable used in the expression and the second one is the SNMP OID which is bound to this variable. This OID is added to the list of OIDs queried during each poll of this server. When a SNMP reply is received, all instances of that variable in the expression are replaced with the value of the corresponding SNMP variable. Once all variables have been thus resolved, the expression is evaluated and its result is taken as the relative load for the given server.

Let's take a more complex example. Suppose you define the relative load to be a function of outgoing data transfer rate through the main network interface and the load average of the server. Data transfer rate is defined as first derivative of data transfer through the interface with respect to time. Let '`out`' be the outgoing data transfer, '`la1`' be the server's 1-minute load

average, '`k`' and '`m`' be two server-dependent constants (*weight coefficients*). Given that, we define relative load as

```
expression "sqrt(k * d(out)**2 + m * la1**2)";
```

The '`**`' operator raises its left operand to the power given by its second operand. The two constructs '`d(...)`' and '`sqrt(...)`' are *function calls*. The '`d()`' function computes first derivative of its argument with respect to time. The '`sqrt()`' function computes the square root of its argument.

Now, let's define variable bindings:

```
variable out "IF-MIB::ifOutOctets.3";
variable la1 "UCD-SNMP-MIB::laLoadFloat.1";
```

The *constants* '`k`' and '`m`' are defined using the following statements:

```
constant k 1.5;
constant m 1.8;
```

The '`constant`' statement is similar to '`variable`', except that its second argument must be a floating-point number. Of course, in this particular example, the two constants could have been placed directly in the expression text, as in:

```
expression "sqrt(1.5 * d(out)**2 + 1.8 * la1**2)";
```

However, defining them on a per-server basis is useful when the same expression is used for several different servers, as explained in the following section.

To conclude, here is our sample server definition:

```
server srv01 {
  host 192.168.10.6;
  community "public";
  expression "sqrt(k * d(out)**2 + m * la1**2)";
  variable out "IF-MIB::ifOutOctets.3";
  variable la1 "UCD-SNMP-MIB::laLoadFloat.1";
  constant k 1.5;
  constant m 1.8;
}
```

3.6 Named Expressions

In most cases, load estimation function is common for all servers (perhaps with server-dependent set of coefficients and/or variable bindings). To simplify configuration, such a common function can be defined once and then used by another '`server`' definitions. It is defined as a *named expression*, i.e. an expression that can be referred to by its name.

Named expressions are defined using the '`expression`' statement outside of '`server`' block. Such statements take two arguments: the name of the expression and its definition, e.g.:

```
expression load "sqrt(k * d(out)**2 + m * la1**2)";
```

Named expressions can be *invoked* from another expressions using the '`@`' operator, followed by expression's name. For example, the following statement in '`server`' block:

```
expression "@load";
```

is in fact equivalent to the expression defined above. The '`@`' construct can, of course, also be used as a term in arithmetic calculations, e.g.:

```
expression "2 * @load + 1";
```

A *default expression* is a named expression which is implicitly applied for '`server`' statements that lack '`expression`' statement. Default expression is defined using the '`default-expression`' statement:

```
default-expression "load";
```

To finish this section, here is an example configuration declaring two servers which use the same default expression, but supply different sets of coefficients:

```
expression load "sqrt(k * d(out)**2 + m * la1**2)";
default-expression "load";

server srv01 {
  host 192.168.10.6;
  community "public";
  variable out "IF-MIB::ifOutOctets.3";
  variable la1 "UCD-SNMP-MIB::laLoadFloat.1";
  constant k 1.5;
  constant m 1.8;
}

server srv02 {
  host 192.168.10.1;
  community "private";
  variable out "IF-MIB::ifOutOctets.3";
  variable la1 "UCD-SNMP-MIB::laLoadFloat.1";
  constant k 2.5;
  constant m 2.2;
}
```

3.7 Preprocessing with m4

In the previous section we have seen how to define two servers which use the same expression to compute relative load. In real configurations, the number of servers is likely to be considerably greater than that, and adding each of them to the configuration soon becomes a tedious and error-prone task. This task is greatly simplified by the use of *preprocessor*. As mentioned earlier, the configuration file is preprocessed using `m4`, a traditional UNIX macro processor. It provides a powerful framework for implementing complex `slb` configurations.

For example, note that in our sample configuration the ‘server’ statements differ by only three values: the server ID, its IP and community. Taking this into account, we may define the following m4 macro:

```
m4_define(‘defsrv’,‘server $1 {
  host $2;
  community "m4_ifelse(‘$3’,,‘public’,‘$3’)";
  variable out "IF-MIB::ifOutOctets.3";
  variable la1 "UCD-SNMP-MIB::laLoadFloat.1";
  constant k 2.5;
  constant m 2.2;
}’)
```

The ‘defsrv’ macro takes two mandatory and one optional argument. Mandatory arguments are the server ID and its IP address. Optional argument is SNMP community; if it is absent, the default community ‘public’ is used.

Notice, that we use m4_define, instead of the familiar define. It is because the default slb setup renames all m4 built-in macro names so they all start with the prefix ‘m4_’. This avoids possible name clashes and makes preprocessor statements clearly visible in the configuration.

Using this new macro, the above configuration is reduced to the following two lines:

```
defsrv(srv01, 192.168.10.6)
defsrv(srv02, 192.168.10.1, private)
```

Declaring a new server is now a matter of adding a single line of text.

The default preprocessor setup defines a set of useful macros, among them m4_foreach (see Section “foreach” in *GNU M4 macro processor*). This macro can be used to further simplify the configuration, as shown in the example below:

```
m4_foreach(args,
 ‘‘srv01, 192.168.10.6’,
  ‘srv02, 192.168.10.1, private’,
  ‘srv03, 192.168.0.3’,
  ‘srv04, 192.168.100.1, foo’,
  ‘srv05, 192.168.100.2, bar’’,
 ‘defserv(args)
’)
```

The second argument to m4_foreach is a comma-separated list of values. The expansion is as follows: for each value from this list, the value is assigned to the first argument (‘args’). Then the third argument is expanded and the result is appended to the overall expansion.

In this particular example, each line produces an expansion of the ‘defserv’ macro with the arguments taken from that line. Note, that each argument in the list must be quoted, because it contains commas. Note also

the use of the optional third arguments to supply community names that differ from the default one.

For a detailed information about `slb` preprocessor feature, see Section 5.1.4 [Preprocessor], page 24.

3.8 Output

When `slb` has finished building load table, it sends it to the output, line by line, using the *output format string*. This format string is similar to the format argument of printf(1): any characters, except '`%`' are copied to the output verbatim; the '`%`' introduces a *conversion specifier*. For example, '`%i`' expands to the ID of the server this line refers to. The '`%w`' specifier expands to the computed relative load for that server, etc. There is a number of such specifiers, they are all described in detail in Section 5.7.1 [output format], page 32.

The default output format is '`%i %w\n`'. This produces a table, each line of which shows a server ID and its relative load. This default is suitable for testing purposes, but for real configurations you will most probably need to create a custom output format. You do so using the '`output-format`' statement:

```
output-format "id=%i host=%h\n";
```

The output format string can be quite complex as shown in the following example:

```
output-format <<EOT
server 10.0.0.1
update add www.example.net 60 IN A %h
send
EOT;
```

This format creates, for each line from the load table, a set of commands for `nsupdate`[1]. The '`<<`' block introduces a *here-document*, a construct similar to that of shell and several other programming languages. It is discussed in [here-document], page 23.

You may need to include server-dependent data in the corresponding output lines. For this purpose, `slb` provides *server macros*. Server macros are special text variables, defined in the '`server`' block, which can be accessed from output format string.

Macros are defined using the '`macro`' statement, whose syntax is similar to that of '`variable`' or '`constant`'. The first argument supplies the name of the macro, and the second one, its contents, or *expansion*, e.g.:

```
server srv01 {
  host 192.168.10.6;
  community "public";
  macro description "some descriptive text";
```

[1] See Section "Dynamic DNS update utility" in *nsupdate(8) man page*.

```
}
```

Macros are accessed using the following conversion specifier:

```
%(name)
```

where *name* is the name of the macro. This specifier is replaced with the actual contents of the macro. The following example uses the '`description`' macro to create a TXT record in the DNS:

```
output-format <<EOT
server 10.0.0.1
update add www.example.net 60 IN A %h
update add %i.example.net 60 IN TXT "%(description)"
send
EOT;
```

Sometimes you may need to produce some output immediately before the load table or after it. Two configuration statements are provided for that purpose: '`begin-output-message`' and '`end-output-message`'. Both take a single string as argument. The string supplied with '`begin-output-message`' is output before formatting the load table, and the one supplied with '`end-output-message`' is output after formatting it.

Continuing our '`nsupdate`' example, the following statement will remove all existing A records from the DNS prior to creating new ones:

```
begin-output-message <<EOT
server 10.0.0.1
update delete www.example.net
send
EOT;
```

You may want to output only a part of the load table. Two statements are provided for this purpose. The '`head` *N*' statement outputs at most *N* entries from the top of the table. The '`tail` *N*' statement outputs at most *N* entries from the bottom of the table. For example, to print only one entry corresponding to the least loaded server, use

```
head 1;
```

By default `slb` prints results to the standard output. To change this, use the '`output-file`' statement. Its argument specifies the name of a file where `slb` output will be directed. If this name starts with a pipe character ('`|`'), `slb` treats the remaining characters as the shell command. This command is executed (using `/bin/sh -c`), and the output is piped to its standard input. Thus, the following statement

```
output-file "| /usr/bin/nsupdate "
            "-k /usr/share/slb/Kvpn.+157+45756.private";
```

directs the formatted output to the standard input of `nsupdate` command.

3.9 Test Mode

The `slb` configuration can be quite complex, therefore it is important to verify that it behaves as expected before actually implementing it in production. Three command line options are provided for that purpose.

The '`--dry-run`' (or '`-n`') option instructs the program to start in foreground mode and print to the standard output what would have otherwise been printed to the output file. Additional debugging information is displayed on the standard error.

The '`--eval`' option initiates expression evaluation mode. In this mode `slb` evaluates the expression whose name is supplied as argument to the option. Actual values of the variables and constants used in the expression are supplied in the command line in form of variable assignments. The result is printed to the standard output. For example, if the configuration file contains

```
expression load "(la1/100 + usr/1024)/2";
```

then the command

```
slb --eval=load la1=30 usr=800
```

will print '`.540625`'.

If the expression contains calls to '`d()`' (derivative), you will need several evaluations to compute its value. The minimal number of evaluations equals the order of the highest derivative computed by the expression, plus 1. Thus, computing the following expression:

```
expression load "sqrt(k * d(out)**2 + m * la1**2)";
```

requires at least two evaluations. To supply several values to a single variable, separate them with commas, e.g.: '`out=16000,20000`'. This will run first evaluation with '`out=16000`' and the second one with '`out=20000`'. For example:

```
$ slb --eval=load k=1.5 m=3 out=16000,20000 la1=0.4
16.3446
```

Notice, that you don't need to supply the same number of values for each variable. If a variable is assigned a single value, this value will be used in all evaluations.

A more elaborate test facility is enabled by the '`--test`' option. This option instructs `slb` to read SNMP variables and their values from a file and act as if they were returned by SNMP. The output is directed to the standard output, unless the '`--output-file`' option is also given.

The input file name is taken from the first non-option argument. If it is '`-`' (a dash) or if no arguments are given, the standard input will be read:

```
slb --test input.slb
```

The input file consists of sections separated by exactly one empty line. A section contains assignments to SNMP variables in the style of `snmpset`[2].

[2] See Section "snmpset" in *snmpset(1) man page*.

Such assignments form several server groups, each group being preceded by a single word on a line, followed by a semicolon. This word indicates the ID of the server to which the data in the group belong. For example:

```
srv01:
UCD-SNMP-MIB::laLoadFloat.1 F 0.080000
IF-MIB::ifOutUcastPkts.3 c 346120120
srv02:
UCD-SNMP-MIB::laLoadFloat.1 F 0.020000
IF-MIB::ifOutUcastPkts.3 c 2357911693
```

This section supplies data for two servers, named ‘`srv01`’ and ‘`srv02`’.

4 SLB Command Line Syntax

The format of `slb` invocation is:

```
slb options args
```

where *options* are command line options and *args* are non-optional arguments.

Non-optional arguments are allowed only if either '`--test`' or '`--eval`' option is used in *options*.

4.1 Program Mode Options

'`--config-file=file`'
'`-c file`' Use *file* instead of the default configuration file.

'`--cron`' Start in *cron* mode. Normally `slb` operates in *daemon* mode, in which it polls the monitored servers at fixed intervals and outputs the resulting load table. In contrast, when in *cron* mode, `slb` performs a single poll, outputs the table and exits. This mode is useful when starting `slb` from cron, hence its name.

Notice that the function '`d`' (derivative, see [derivative], page 29) does not work in this mode.

'`--dry-run`'
'`-n`' Run in foreground mode and print to the standard output what would have otherwise been printed to the output file. This option implies '`--stderr --debug snmp.1 --debug output.1`'. Use additional '`--debug`' options to get even more info.

The '`pidfile`' configuration statement is ignored in dry run mode (see [pidfile], page 26).

'`--eval=name`'
Evaluate the named expression *name*, print its result and exit. Arguments for the expression can be supplied in the form of assignments in the command line, e.g.:

```
slb --eval=loadavg la1=10 x=18
```

See Section 3.9 [Test Mode], page 14, for a detailed discussion of the expression evaluation mode.

'`--test`' Test mode. Instead of polling servers via SNMP, `slb` reads data from the file given as the first non-option argument on the command line (or from the standard input, if no arguments are given). The output is directed to the standard output, unless the '`--output-file`' option is also given.

See Section 3.9 [Test Mode], page 14, for a detailed information about the test mode, including a description of the input format.

Example usage:

```
slb --test input.slb
```

‘`-E`’ Show preprocessed configuration and exit.

‘`--lint`’
‘`-t`’ Parse configuration file, report any errors on the standard error and exit with code 0, if the syntax is OK, and with code 1 otherwise.

4.2 Modifier Options

‘`--foreground`’
Remain in the foreground. Useful for debugging purposes. See [foreground statement], page 26.

‘`--output-file=`*file*’
‘`-o` *file*’ Direct output to *file*. This option overrides the ‘`output-file`’ setting in the configuration file. See [output-file], page 32, for a discussion of *file* syntax.

4.3 Logging Control Options

‘`--stderr`’
‘`-e`’ Log to the standard error.

‘`--syslog`’
Log all diagnostics to syslog.

4.4 Preprocessor Control Options

‘`--define=`*name*`[=`*value*`]`’
‘`-D`*name*`[=`*value*`]`’
Define the preprocessor symbol *name* as having *value*, or empty. See Section 5.1.4 [Preprocessor], page 24.

‘`--include-directory=`*dir*’
‘`-I` *dir*’ Add *dir* to include search path.

See Section 5.1.2 [Pragmatic Comments], page 21.

‘`--no-preprocessor`’
Disable preprocessor. see Section 5.1.4 [Preprocessor], page 24.

‘`--preprocessor=`*command*’
Use *command* instead of the default preprocessor. see Section 5.1.4 [Preprocessor], page 24.

4.5 Debugging Options

‘`--debug=`*cat*`[.`*level*`]`’
‘`-d` *cat*`[.`*level*`]`’
Sets debugging level for the category *cat*.

Category is a string that designates a part of the program from which the additional debugging information is requested. See below for a list of available categories.

Level is a decimal number between 0 and 100 which indicates how much additional information is required. The level of 0 means ‘`no information`’ and effectively disables the category in question. The level of 100 means maximum amount of information available. If *level* is omitted, 100 is assumed.

The following table lists categories available in version 1.0:

main	Main program block.
eval	Expression evaluation.
egram	Expression grammar and parser.
elex	Expression lexical analyzer.
snmp	SNMP request-reply loop.
output	Output driver.
cfgram	Configuration file grammar.
cflex	Configuration file lexer.

4.6 Informational Options

‘`--config-help`’
Show configuration syntax summary.

‘`--help`’
‘`-h`’ Print a concise usage summary and exit.

‘`--usage`’ Print a summary of command line syntax and exit.

‘`--version`’
‘`-v`’ Print the program version and exit.

5 SLB Configuration File

Upon startup, `slb` reads its settings from the *configuration file* '`slb.conf`'. This file is normally located in *$sysconfidr* (i.e., in most cases '`/usr/local/etc`', or '`/etc`'), but an alternative location can be specified using the '`--config`' command line option (see Section 3.2 [Configuration Basics], page 5).

If any errors are encountered in the configuration file, the program reports them on its error output and exits with a non-zero status.

To test the configuration file without starting the server use the '`--lint`' ('`-t`') command line option. It instructs the program to check configuration file for syntax errors and other inconsistencies and to exit with status 0 if no errors were detected, and with status 1 otherwise.

Before parsing, the configuration file is preprocessed using `m4` (see Section 5.1.4 [Preprocessor], page 24). To see the preprocessed configuration without actually parsing it, use the '`-E`' command line option. To disable preprocessing, use the '`--no-preprocessor`' option.

The rest of this section describes the configuration file syntax in detail. You can receive a concise summary of all configuration directives any time by running `slb --config-help`.

5.1 Configuration file syntax

The configuration file consists of statements and comments.

There are three classes of lexical tokens: keywords, values, and separators. Blanks, tabs, newlines and comments, collectively called *white space* are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent keywords and values.

5.1.1 Comments

Comments may appear anywhere where white space may appear in the configuration file. There are two kinds of comments: single-line and multi-line comments. *Single-line* comments start with '`#`' or '`//`' and continue to the end of the line:

```
# This is a comment
// This too is a comment
```

Multi-line or *C-style* comments start with the two characters '`/*`' (slash, star) and continue until the first occurrence of '`*/`' (star, slash).

Multi-line comments cannot be nested. However, single-line comments may well appear within multi-line ones.

5.1.2 Pragmatic Comments

Pragmatic comments are similar to usual single-line comments, except that they cause some changes in the way the configuration is parsed. Pragmatic

comments begin with a '`#`' sign and end with the next physical newline character. The version 1.0 of `slb`, understands the following pragmatic comments:

`#include <`*file*`>`
`#include` *file*
: Include the contents of the file *file*. If *file* is an absolute file name, both forms are equivalent. Otherwise, the form with angle brackets searches for the file in the *include search path*, while the second one looks for it in the current working directory first, and, if not found there, in the include search path.

  The default include search path is:

  1. '*prefix*`/share/slb/1.0/include`'
  2. '*prefix*`/share/slb/include`'

  where *prefix* is the installation prefix.

  New directories can be appended in front of it using '`-I`' ('`--include-directory`') command line option (see Section 5.1.4 [Preprocessor], page 24).

`#include_once <`*file*`>`
`#include_once` *file*
: Same as `#include`, except that, if the *file* has already been included, it will not be included again.

`#line` *num*
`#line` *num* `"`*file*`"`
: This line causes `slb` to believe, for purposes of error diagnostics, that the line number of the next source line is given by *num* and the current input file is named by *file*. If the latter is absent, the remembered file name does not change.

`#` *num* `"`*file*`"`
: This is a special form of `#line` statement, understood for compatibility with the C preprocessor.

In fact, these statements provide a rudimentary preprocessing features. For more sophisticated ways to modify configuration before parsing, see Section 5.1.4 [Preprocessor], page 24.

5.1.3 Statements

A *simple statement* consists of a keyword and value separated by any amount of whitespace. Simple statement is terminated with a semicolon ('`;`').

The following is a simple statement:

```
standalone yes;
pidfile /var/run/slb.pid;
```

A *keyword* begins with a letter and may contain letters, decimal digits, underscores ('`_`') and dashes ('`-`'). Examples of keywords are: '`expression`', '`output-file`'.

A *value* can be one of the following:

number
: A number is a sequence of decimal digits.

boolean
: A boolean value is one of the following: '`yes`', '`true`', '`t`' or '`1`', meaning *true*, and '`no`', '`false`', '`nil`', '`0`' meaning *false*.

unquoted string
: An unquoted string may contain letters, digits, and any of the following characters: '`_`', '`-`', '`.`', '`/`', '`@`', '`*`', '`:`'.

quoted string
: A quoted string is any sequence of characters enclosed in double-quotes ('`"`'). A backslash appearing within a quoted string introduces an *escape sequence*, which is replaced with a single character according to the following rules:

Sequence	Replaced with
\a	Audible bell character (ASCII 7)
\b	Backspace character (ASCII 8)
\f	Form-feed character (ASCII 12)
\n	Newline character (ASCII 10)
\r	Carriage return character (ASCII 13)
\t	Horizontal tabulation character (ASCII 9)
\v	Vertical tabulation character (ASCII 11)
\\	A single backslash ('\')
\"	A double-quote.

Table 5.1: Backslash escapes

In addition, the sequence '`\`*`newline`*' is removed from the string. This allows to split long strings over several physical lines, e.g.:

```
"a long string may be\
 split over several lines"
```

If the character following a backslash is not one of those specified above, the backslash is ignored and a warning is issued.

Two or more adjacent quoted strings are concatenated, which gives another way to split long strings over several lines to improve readability. The following fragment produces the same result as the example above:

```
"a long string may be"
" split over several lines"
```

Here-document
: A *here-document* is a special construct that allows to introduce strings of text containing embedded newlines.

The `<<word` construct instructs the parser to read all the following lines up to the line containing only *word*, with possible trailing blanks. Any lines thus read are concatenated together into a single string. For example:

```
<<EOT
A multiline
string
EOT
```

The body of a here-document is interpreted the same way as a double-quoted string, unless *word* is preceded by a backslash (e.g. '`<<\EOT`') or enclosed in double-quotes, in which case the text is read as is, without interpretation of escape sequences.

If *word* is prefixed with `-` (a dash), then all leading tab characters are stripped from input lines and the line containing *word*. Furthermore, if `-` is followed by a single space, all leading whitespace is stripped from them. This allows to indent here-documents in a natural fashion. For example:

```
<<- TEXT
    The leading whitespace will be
    ignored when reading these lines.
TEXT
```

It is important that the terminating delimiter be the only token on its line. The only exception to this rule is allowed if a here-document appears as the last element of a statement. In this case a semicolon can be placed on the same line with its terminating delimiter, as in:

```
help-text <<-EOT
        A sample help text.
EOT;
```

A *block statement* introduces a logical group of statements. It consists of a keyword, followed by an optional value, and a sequence of statements enclosed in curly braces, as shown in the example below:

```
server srv1 {
  host 10.0.0.1;
  community "foo";
}
```

The closing curly brace may be followed by a semicolon, although this is not required.

5.1.4 Preprocessor

Before actual parsing, the configuration file is preprocessed. The built-in preprocessor handles only file inclusion and `#line` statements (see Section 5.1.2 [Pragmatic Comments], page 21), while the rest of traditional preprocessing facilities, such as macro expansion, is supported via `m4`, which serves as external preprocessor.

The detailed description of `m4` facilities lies far beyond the scope of this document. You will find a complete user manual in Section "GNU M4" in *GNU M4 macro processor*. For the rest of this subsection we assume the reader is sufficiently acquainted with `m4` macro processor.

The external preprocessor is invoked with '`-s`' flag, which instructs it to include line synchronization information in its output. This information is then used by the parser to display meaningful diagnostic.

An initial set of macro definitions is supplied by the '`pp-setup`' file, located in '*$prefix*`/share/slb/`*version*`/include`' directory (where *version* means the version of SLB).

The default '`pp-setup`' file renames all `m4` built-in macro names so they all start with the prefix '`m4_`'. This is similar to GNU m4 '`--prefix-builtin`' options, but has an advantage that it works with non-GNU `m4` implementations as well.

Additional control over the preprocessor is provided via the following command line options:

'`--define=`*name*`[=`*value*`]`'
'`-D`*name*`[=`*value*`]`'
: Define the preprocessor symbol *name* as having *value*, or empty.

'`--include-directory=`*dir*'
'`-I`*dir*'
: Add *dir* to the list of directories searched for preprocessor include files.

'`--no-preprocessor`'
: Disable preprocessor.

'`--preprocessor=`*command*'
: Use *command* instead of the default preprocessor.

5.2 Syslog Configuration Directives

When running in standalone mode `slb` normally uses `syslog` to print diagnostic messages. By default, the program uses the '`daemon`' facility. The `syslog` statement allows to change that:

`syslog` *{ ... }* [Config]

```
syslog {
  facility local1;
  tag slb;
  print-priority yes;
}
```

`facility` *name* [Config: syslog]

Configures the syslog facility to use. Allowed values are: '`auth`', '`authpriv`', '`cron`', '`daemon`', '`ftp`', '`local0`' through '`local7`', and '`mail`'.

`tag` *string* [Config: syslog]
This statement sets the *syslog tag*, a string identifying each message issued by the program. By default, it is the name of the program with the directory parts removed.

`print-priority` *bool* [Config: syslog]
In addition to priority segregation, provided by `syslog`, you can instruct `slb` to prefix each syslog message with its priority. To do so, set:

```
print-priority yes;
```

5.3 Daemon Configuration

The following statements configure `slb` activities in daemon mode.

`standalone` *bool* [Config]
Enables or disables standalone daemon mode. The standalone mode is enabled by default. It is disabled either by setting

```
standalone no;
```

in the configuration file, or by using the '`--cron`' command line option (see [option–cron], page 17).

`foreground` *bool* [Config]
Do not detach from the controlling terminal. See [option–foreground], page 18.

`pidfile` *string* [Config]
Store master process PID in *file*. Default pidfile location is '`/var/run/slb.pid`'.

`wakeup` *number* [Config]
Sets wake-up interval in seconds. SLB will recompute the server load table each *number* seconds.

`suppress-output` *number* [Config]
Suppress output during the first *number* wakeups. This statement is reserved mostly for debugging purposes.

5.4 Expression

The following statement creates a *named expression*:

`expression` *name expr* [Config]
Define the expression *name* to be *expr*.

Named expressions can be used as load estimation functions in `server` statements (see Section 5.6 [server], page 30) and invoked from the output format string (see Section 5.7.1 [output format], page 32).

`default-expression` *name* [Config]
Declares the *default expression*. The *name* refers to a named expression declared elsewhere in the configuration file.

The default expression is used to compute relative load of servers whose '`server`' declaration lacks explicit '`expression`' definition (see [server-expression], page 31).

The *expr* in the '`expression`' statement is an arithmetical expression, which evaluates to a floating point number. The expression consists of *terms*, *function calls* and operators. The *terms* are floating point numbers, variable and constant names. The names refer to constants or SNMP variables defined in the definition of the server, for which this expression is being evaluated.

The following operations are allowed in expression:

Arithmetic operations

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`**`	Power

Named expression reference

A *named expression reference* is a reference to an expression defined by a '`expression`' statement elsewhere in the configuration file. The reference has the following syntax:

```
@name
```

where *name* is the name of the expression as given by the first argument of its definition (see Section 5.4 [expression], page 26).

Ternary conditional operator

The *ternary conditional operator* is used to select a value based on a condition. It has the following form:

```
cond ? expr1 : expr2
```

The ternary operator evaluates to *expr1* if *cond* yields '`true`' (i.e. returns a non-null value) and to *expr2* otherwise.

The condition *cond* is an expression which, apart from the arithmetic operators above, can use the following *comparison* and *logical* operations:

5.4.1 Comparisons

$a == b$	True if a equals b

a != *b*	True if the operands are not equal.
a < *b*	True if *a* is less than *b*.
a <= *b*	True if *a* is less than or equal to *b*.
a > *b*	True if *a* is greater than *b*.
a >= *b*	True if *a* is greater than or equal to *b*.

Logical operations

!*expr*	True if *expr* is false.
a && *b*	Logical *and*: true if both *a* and *b* are true, false otherwise.
a || *b*	Logical *or*: true if at least one of *a* or *b* are true.

Both logical '`and`' and '`or`' implement *boolean shortcut evaluation*: their second argument (*b*) is evaluated only when the first one is not sufficient to compute the result.

Precedence

The following table lists all operators in order of decreasing precedence:

Operators	**Description**
(...)	Grouping
?	Ternary operator
**	Power (right-associative)
-	Unary negation
* /	Multiplication, division
+ -	Addition, subtraction
< <= >= >	Relational operators (non-associative)
== !=	Equality comparison (non-associative)
!	Boolean negation
&&	Logical '`and`'.
||	Logical '`or`'

When operators of equal precedence are used together they are evaluated from left to right (i.e., they are *left-associative*), except for comparison operators, which are non-associative and for the power operator, which is right-associative (these are explicitly marked as such in the above table). This means, for example that you cannot write:

```
(5 <= x <= 10) ? x : y
```

Instead, you should write:

```
(5 <= x && x <= 10) ? x : y
```

5.4.2 Function calls

Function calls have the following syntax:

```
name(arglist)
```

where *name* stands for the function name and *arglist* denotes the *argument list*: a comma-separated list of expressions which are evaluated and supply actual arguments for the function.

The following functions are supported by SLB version 1.0:

`d` (*x*) [function]
Returns the derivative of *x*, i.e. the speed of its change per second, measured between the two last wakeups of `slb` (see Section 3.3 [Daemon Configuration], page 7).

Notice, that this function is available only in standalone mode (see Section 3.3 [Daemon Configuration], page 7).

`max` (*x0*, ..., *xn*) [function]
Returns the maximum value of its arguments. Any number of arguments can be given.

`min` (*x0*, ..., *xn*) [function]
Returns the minimum value of its arguments.

`avg` (*x0*, ..., *xn*) [function]
Returns the average value of its arguments.

`log` (*x*) [function]
Returns the natural logarithm of *x*.

`log10` (*x*) [function]
Returns the decimal logarithm of *x*.

`exp` (*x*) [function]
Returns the value of *e* (the base of natural logarithms) raised to the power of *x*.

`pow` (*x*, *y*) [function]
Returns the value of *x* raised to the power of *y*. This function is provided for the sake of completeness, as it is entirely equivalent to '`x ** y`'.

`sqrt` (*x*) [function]
Returns the non-negative square root of *x*.

`abs` (*x*) [function]
Returns the absolute value of *x*.

`ceil` (*x*) [function]
Returns the smallest integral value that is not less than *x*.

```
ceil(0.5) ⇒ 1.0
ceil(-0.5) ⇒ 0
```

`floor` (*x*) [function]
Returns the largest integral value that is not greater than *x*.

```
floor(0.5) ⇒ 0.0
floor(-0.5) ⇒ -1.0
```

`trunc` (*x*) [function]
Rounds *x* to the nearest integer not larger in absolute value.

`round` (*x*) [function]
Rounds *x* to the nearest integer, but round halfway cases away from zero:

```
round(0.5) ⇒ 1.0
round(-0.5) ⇒ -1.0
```

5.5 SNMP Configuration

`mib-directory` *dir* [Config]
Adds *dir* to the list of directories searched for the MIB definition files.

`add-mib` *file* [Config]
Reads MIB definitions from *file*.

5.6 Server Configuration

`server` *id { ... }* [Config]

```
server id {
  enable bool;
  host string;
  timeout number;
  retries number;
  community string;
  expression expr;
  variable name oid;
  constant name value;
  macro name expansion;
  assert oid op pattern;
}
```

Creates a new entry in the database of monitored servers. The *id* parameter supplies the *server identifier*, an arbitrary string which will be used in log messages related to that server. Its value is also available in output format string as format specifier '`%i`' (see Section 5.7.1 [output format], page 32).

Other parameters of the server are defined in the substatements, as described below.

`enable` *bool* [Config: server]
If *bool* is '`no`', this server is disabled, i.e. it is not polled nor taken account of in load calculations. Use this statement if you want to temporarily disable a server.

`host` *string* [Config: server]
Sets the IP address (or hostname) of this server.

`timeout` *number* [Config: server]
Sets the timeout for SNMP requests to that server.

`retries` *number* [Config: server]
Sets the maximum number of retries for that server, after which a timeout is reported.

`community` *string* [Config: server]
Sets SNMP community.

`expression` *string* [Config: server]
Defines load estimation function for this server. See Section 5.4 [expression], page 26, for a description of the syntax of *string*. If this statement is absent, the default expression will be used (see [default-expression], page 26). If it is not declared as well, an error is reported.

`variable` *name oid* [Config: server]
Defines a variable *name* to have the value returned by *oid*. The latter must return a numeric value.

Any occurrence of *name* in the expression (as defined in the '`expression`' statement) is replaced with this value.

`constant` *name number* [Config: server]
Defines a constant *name* to have the value returned by *oid*.

Notice that expression variables and constants share the same namespace, therefore it is an error to have both a '`variable`' and a '`constant`' statement defining the same *name*.

`macro` *name expansion* [Config: server]
Defines a macro *name* to expand to *expansion*. Macros allow to customize output formats on a per-server basis. See [macro], page 12.

`assert` *oid op pattern* [Config: server]
Ensures that the value of SNMP variable *oid* matches *pattern*. The type of match is given by the *op* argument:

eq The value must match *pattern* exactly.

ne The value must not match *pattern*.

If the assertion fails, the server is excluded from the load table.

Use this statement to ensure that a variable used in the computation refers to a correct entity. For example, if your expression refers to '`IF-MIB::ifInOctets.3`' (number of input octets on network interface 3), it would be wise to ensure that the 3rd row refers to the interface in question (say '`eth1`'):

```
assert "IF-MIB::ifDescr.3" eq "STRING: eth1";
```

Notice, that *pattern* must include the data type.

5.7 Output Configuration

The statement discussed in this section configure formatting of the server load table on output.

`head` *number* [Config]
Print at most *number* entries from the top of the table, i.e. the *number* of the less loaded server entries.

`tail` *number* [Config]
Print at most *number* entries from the bottom of the table, i.e. the *number* of the most loaded server entries.

`output-file` *string* [Config]
Send output to the channel, identified by *string*. Unless *string* starts with a pipe sign, it is taken as a literal name of the file. If this file does not exist, it is created. If it exists, it is truncated. The file name can refer to a regular file, symbolic link or named pipe.

If *string* starts with a '`|`' (pipe), the rest of the string is taken as the name of an external program and its command line arguments. The program is started before `slb` enters its main loop and the formatted load table is piped on its standard input.

`begin-output-message` *string* [Config]
Defines the text to be output before the actual load table contents. The *string* is taken literally. For example, if you want it to appear on a line by itself, put '`\n`' at the end of it (see Table 5.1).

`end-output-message` *string* [Config]
Defines the text to be output after the actual load table contents. The *string* is taken literally.

`output-format` *string* [Config]
Defines *format* for outputting load table entries. See the subsection below (see Section 5.7.1 [output format], page 32), for a description of available format specifications.

5.7.1 Output Format String

The format string is composed of zero or more directives: ordinary characters (not '`%`'), which are copied unchanged to the output; and *conversion specifications*, each of which is replaced with the result of a corresponding conversion. Each conversion specification is introduced by the character '`%`', and ends with a *conversion specifier*. In between there may be (in this order) zero or more *flags*, an optional *minimum field width* and an optional *precision*.

Flags

0 The value should be zero padded. This affects all floating-point conversions, i.e. '`w`' and '`{...}`', for which the resulting value is padded on the left with zeros rather than blanks. If the '`0`' and '`-`' flags both appear, the '`0`' flag is ignored. If a precision is given with a floating-point conversion, the '`0`' flag is ignored. For other conversions, the behavior is undefined.

- The converted value is to be left adjusted on the field boundary. (The default is right justification.) Normally, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A '`-`' overrides a '`0`' if both are given.

' ' (a space)
A blank should be left before a positive number (or empty string) produced by conversion.

The field width

An optional decimal digit string (with non-zero first digit) specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the '`-`' flag has been given).

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

The precision

An optional precision, in the form of a period ('`.`') followed by an optional decimal digit string. If the precision is given as just '`.`', or the precision is negative, the precision is taken to be zero. This gives the number of digits to appear after the radix character for floating-point conversions or the maximum number of characters to be printed from a string, for string conversions.

The conversion specifier

A character or a sequence of characters that specifies the type of conversion to be applied. The conversion specifiers are replaced in the resulting output string as described in the table below:

i The server identifier.

h The server hostname, as set by the `host` configuration directive.

w Computed relative load.

{*var*} The value of the variable *var*.

{@*expr-name*}
The result of evaluating expression *expr-name* in the context of the current server.

(*macro*) Expansion of the *macro*.

% The percent character.

6 Exit Codes

When SLB terminates, it reports the result of its invocation in form of exit code. Exit code of 0 indicates normal termination. Non-zero exit codes indicate some kind of error. In this case the exact cause of failure will be reported on the currently selected logging channel.

The exit codes are as follows:

0 (EX_OK)
: Normal termination.

64 (EX_USAGE)
: The program was invoked incorrectly.

65 (EX_DATAERR)
: Input data were invalid or malformed. This error code is returned only when `slb` is used with '`--test`' or '`--eval`' options (see Section 3.9 [Test Mode], page 14).

69 (EX_UNAVAILABLE)
: Some error occurred. For example, the program was unable to open output file, etc.

70 (EX_SOFTWARE)
: Internal software error. This usually means hitting a bug in the program, so please report it (see Chapter 8 [Reporting Bugs], page 39).

98 (EX_CONFIG)
: Program terminated due to errors in configuration file.

7 Signals

The program handles a set of signals. To send a signal to `slb` use the following command:

```
kill -signal `cat /var/run/slb.pid`
```

Replace '`/var/run/slb.pid`' with the actual pathname of the PID-file, if it was set using the '`pidfile`' configuration statement.

The '`SIGHUP`' signal instructs the running instance of the program to restart itself. It is possible only if `slb` was started using its full pathname.

The signals '`SIGTERM`', '`SIGQUIT`', and '`SIGINT`' cause immediate termination of the program.

8 How to Report a Bug

Please, report bugs and suggestions to gray+slb@gnu.org.ua.

You hit a bug if at least one of the conditions below is met:

- `slb` terminates on signal 11 (SIGSEGV) or 6 (SIGABRT).
- `slb` terminates with exit code 70 (internal software error)
- The program fails to do its job as described in this manual.

If you think you've found a bug, please be sure to include maximum information available to reliably reproduce it, or at least to analyze it. The information needed is:

- Version of the package you are using.
- Compilation options used when configuring the package.
- Command line options used.
- Configuration file.
- Conditions under which the bug appears.

Any errors, typos or omissions found in this manual also qualify as bugs. Please report them, if you happen to find any.

Appendix A GNU Free Documentation License

Version 1.2, November 2002

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 noncommercially. 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.

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 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 for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

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.

A.1 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 1.2 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

This is a general index of all issues discussed in this manual

I

L

M

N

O

P

Q

R

S

T

U

V

W