Logging in C++ with google logging library

Info: See http://google-glog.googlecode.com
Author: Chad Skeeters, Jerry Garcia
Date: 13-Feb-2013
Description: This document outlines the use of google logging library

ABSTRACT

This document describes the facilities in the google logging library that may be used for a C++ project.

Using a logging library can assist developers with debugging code and help maintenance personnel with troubleshooting systemic problems. C++ developers often use `printf` or `cout` statements to check for proper code flow and to debug issues, but these statements must later be removed and can become tedious to manage or adjust over time. Many logging facilities feature helpful *logging levels* that allow the user to control logging at a specified level or above, all without the need for re-compilation.

Source Code Example

The following example shows how the library is used.

```cpp
#include <glog/logging.h>

int main(int argc, char *argv[])
{
  // Flags can control which logs are output.
  // Many flags can be changed after InitGoogleLogging has been called

  //FLAGS_stderrthreshold=google::INFO;
  //FLAGS_logtostderr = 1; // disabled logging to a file.
  //FLAGS_minloglevel = google::WARNING; // the default is INFO
  //FLAGS_v = 2;
  google::InitGoogleLogging(argv[0]);

  // Unlike cout, log statements don't require std::endl
  LOG(FATAL) << "Program is dying because...";
  LOG(ERROR) << "Error has occurred, but trying to continue";
  LOG(WARNING) << "Warning something might not be set up right.";
  LOG(INFO) << "Something is happening.";

  // Verbose Logging for tracing execution
  VLOG(1) << "Something is happening";
  VLOG(2) << "Something detailed is happening";
  VLOG(3) << "Something very detailed is happening";

}
```

General Process

When a logging statement is executed in the program, the level of the log (`FATAL`, `ERROR`, ...) specified in the statement is checked against a minimum logging level that is configured in the library to see if the statement should be ignored. If unfiltered, the string will get output to stderr or a log file, or both.

Logging Types

Generally, the most useful logs for administrators and users are those with INFO, WARNING, ERROR, and FATAL levels.

Verbose Logging are logs issued with the `VLOG` keyword. Developers and Maintainers may find the *Verbose Logging* capabilities useful for tracking program execution. Levels for verbose logging are numbers (instead of predefined words) starting with level 1. Larger numbers trigger all of the verbose logging with a level number less than or equal to the level specified.

Output

Calling `google::InitGoogleLogging(argv[0])` inside your program will enable log output to a file in the `/tmp/` directory. This does not prevent the logs from being output on `stderr` as well.

The name of the program (passed in `argv[0]`) is used as a part of the file name of the log files that are created in `/tmp/`. Unless otherwise specified, glog writes to the filename:

```
/tmp/<program name>.<hostname>.<user name>.log.<severity level>.<date>.<time>.<pid>
(e.g., "/tmp/hello_world.server.chad.log.INFO.20080709-222411.10474")
```

In addition to these log files, symbolic links are also generated that make it easier for the user to see the most recently generated logs. In the above example, the symbolic link would be named `/tmp/hello_world.INFO`.

Configuring

There are a number of ways in which the logging settings may be configured. Logging settings can be configured by either:

- Setting flags inside the source code (see the commented lines in the previous source code example)
- Providing command line arguments to the application that will set the flags (See google flags)
- Setting environment variables before the program starts

Most flags documented as a command line parameter can alternatively be used as a program variable (with the prefix `FLAGS_`) or as an environment variable (with the prefix `GLOG_`).

> Note
>
> The command line arguments don't work out of the box without google flags, or some other command line argument processor.

Standard Output

By default, logs with level `ERROR` and `FATAL` will print to stderr in addition to being output to the logfile. This default level may be changed by setting the `FLAGS_stderrthreshold` variable either to 0 for `INFO`, 1 for `WARNING`, 2 for `ERROR`, or 3 for `FATAL`. The `GLOG_stderrthreshold` environment variable can be used the same way as demonstrated in the following example:

```
GLOG_stderrthreshold=0 ./hello_world
```

Log Levels

An administrator may decide that they don't want to clutter their files with too many INFO level logs. This can be adjusted by changing the logging levels. To change the logging level, run the program with different values for the environment variable `GLOG_minloglevel` set to 0 for `INFO`, 1 for `WARNING`, 2 for `ERROR`, and 3 for `FATAL`. Since it's a minimum, setting it to 1 for `WARNING` would include `ERROR` and `FATAL`, but exclude `INFO`.

```
GLOG_minloglevel=1 ./hello_world
```

Looking again at the source code example at the beginning of this document, if the line with the FLAGS_minloglevel flag is uncommented, a default will be set that can be overridden by an argument or environment variable.

Verbose Log Levels

Verbose logging can be enabled globally via the `FLAGS_v` flag or the `GLOG_v` environment variable. It can also be enabled on a per-module basis using the `GLOG_vmodule` environment variable.

```
GLOG_vmodule=processor=2 GLOG_v=1 ./hello_world
```

The above example enables any `VLOG(2)` statements in `processor.h` or `processor.cpp`.

Installation

The google logging library and development header can be downloaded on the google code website.

Please read the INSTALL file for details. The following command should work to install:

```
sudo make install
```

Note

There are also RPM files on google code if you look for outdated files.

Compiling your programs

When linking your program you must pass the `-lglog` flag. You can alternatively use the following:

```
pkg-config --libs libglog
```