A Common Logging Framework (FhgLog)

This page describes a newly developed logging framework which can be used from within SDPA, Gwes and other projects.

Requirements analysis

What should the framework provide and how does one want to use it.

- completely removable at compile-time
- fast and easy to use

What should be available for logging statements:

- line number
- file name (full path)
- short file name
- function name
- thread identifier
- log message
- in case of over-the-net logging, hostname and pid should be included as well
  - the configured time may differ between log-host and the host emitting the log event!

Different logging destinations:

- it should be possible to log to various destinations including:
  - console output
  - log file
  - network
  - extendable by the client library/application

Currently implemented

- line number
- path, filename
- function (short, long)
- process id, thread id
- log message
- console/stream/file output
- custom formatter with string-fmt parsing
- timestamp (microseconds, ascii)
- remote logging
  - configure FhgLog? with -DWITH_REMOTE_LOGGING=Yes and the remote logging files will be built
  - libfhglog_remote
  - new Appender fhglog/remote/RemoteAppender.hpp
  - binaries fhglogd and fhglogc (the latter can be used to log from within scripts)
  - default port: UDP 2438
- colored output if destination is a tty
- environment configurable:

    - FHGLOG_level={MIN,MAX,TRACE,INFO,WARN,ERROR,FATAL}, whereas MIN==TRACE, MAX==FATAL
    - FHGLOG_to_server=host[:port]
    - FHGLOG_to_file=path
    - FHGLOG_color={on,off,auto}
    - FHGLOG_format="formatstring"
    - FHGLOG_to_console="stderr,stdout,stdlog"
- FATAL messages are handled especially:
    - by default the program aborts
    - the user has the possibility to modify this behavior by registering a handler function
- command line client that can be used similar to 'logger' from syslog

Usage scenarios

- log to console
- log to network
- log to syslog
- see the various test cases for example code

setting up and using the logging

```
#include <fhglog/minimal.hpp>

int main(int ac, char *av[])
{
  FHGLOG_SETUP(ac,av);
  // or just
  FHGLOG_SETUP();

  // regular logging
  LOG(<level>, "hello" << " " << "world!");
  LOG_IF(<level>, <cond>, <msg>);

  // debug build logging (if NDEBUG is *not* defined)
  DLOG(...);
  DLOG_IF(...);

  return 0;
}
```

```
#include <fhglog/fhglog.hpp>
#include <fhglog/Configuration.hpp>
#include <fhglog/remote/RemoteAppender.hpp>
using namespace fhg::log;

int main (int, char**)
{
   // default configuration
   fhg::log::Configurator::configure();

   // manual configuration
   logger_t root(getLogger());
   root.addAppender(Appender::ptr_t(new StreamAppender("console")))->set
   root.addAppender(Appender::ptr_t(new StreamAppender("console")))->set
   root.addAppender(Appender::ptr_t(new remote::RemoteAppender("remote",

   logger_t log(getLogger("my.module"));
   log.setLevel(LogLevel::INFO);
   FHGLOG_DEBUG(root, "test message " << 42);
   FHGLOG_INFO(log, "test message " << 42);

   // additional macros
   LOG(DEBUG, "message"); // logs to the default (i.e. root) logger
   DLOG(INFO, "message"); // logs only if NDEBUG has not been defined

   MLOG(DEBUG, "message"); // same as above, but logs to a "file-local" l
                           // the logger's name is constructed from the f
   DMLOG(INFO, "message"); // same as above
   return 0;
}
```