Hielog Library

TODO: Change namespace name and copyright year along the entire source code.

TODO: Redesign Providers.

TODO: Add Thread storage cleanup tool.

TODO: Singleton example.

TODO: DateTime example.

TODO: Custom Provider example.

TODO: Custom Formatter example.

TODO: Custom String Formatter extension.

TODO: Custom Log Message example.

TODO: Check all code snippets here

HieLog (Hierarchical Logging) is a lightweight extensible tool for general-purpose logging.

Introduction

HieLog library is designed to stream log statements and maintain hierarchy of calls automatically. Core features of the library are ability to provide logging facilities for concurrent enabled applications and a chance to stream log over the network.

Rationale

There are various applications of logging. Most of the related tasks are associated with logging at development/debug time.

HieLog addresses the following goals:

- **Scalability.** Concurrency is integral component of modern software.
- **Extensibility.** Logging itself is too wide area to be covered by a single tool. The library is intended to provide skeleton for building complex, production level solutions. However, it includes basic functionality to start using it right away.
- **Portability.** Including mobile devices.
- **Ease of use.** HieLog is implemented as header library to allow low cost integration. Its syntax is clear and simple. All potential use cases are covered in Tutorial and Samples sections.

Supported Platforms

HieLog is distributed as header library and depends only on tools from Boost (see Dependencies section).

Dependencies

Library itself depends on just few STD headers (like <iostream>, <iterator>, <string> and <fstream>). Its major core components depend on the following Boost libraries:

- Add noncopyable add smart ptrs
- Boost.Thread for concurrent logging.
- Boost.Asio for network enabled logging.
- Boost.Interprocess for process id information.
- Boost.Date_Time for extra logging details.

Some of libraries (for example Boost.Date_Time) require building Boost, so there is a chance to turn corresponding functionality off with the help of one of the following macros (see hielog/config.hpp for the entire set of configuration options):

Macro	Description
HIELOG_NO_THREADS	Explicitly disables concurrent logging support.
HIELOG_NO_NETWORK	Explicitly disables networking support.
HIELOG_NO_DATETIME	Explicitly disables embedding timings in log statements.

Concepts

Logging is provided by means of hielog::Log class. This class should be parameterized with two arguments, namely log provider and log formatter. These are major concepts of HieLog library. *Provider* provides buffer to stream log messages to and is responsible for indentation management. *Formatter* handles each message and dumps it into string.

```
using namespace hielog;
typedef log< basic_provider, basic_formatter > log_type;
log_type log_instance;
```

Here comes major extensibility advantage of HieLog. Log object (as well as almost all providers and formatters from the original package) is configurable.

It is allowed to instantiate as many instances of hielog::Log class as necessary. It is also up to the caller to share these instances among the application. It was initially decided to avoid embedding singletons. However there is an example of doing this below.

There are two ways of putting log messages. The first one is called ‘log scope’ and is intended to put message at the beginning and ending of C++ scope (code fragment wrapped by braces).

```
{
    log_scope< log_type > scope(log_instance, “Scope”);
    // remaining scope code
}
```

The statement above puts heading message and increases indent for all messages sent from within the scope.

The second method (‘log message’) allows putting single message to log stream.

```
log_message< log_type > message(log_instance, "Message");
```

Current implementation of scope and message supports long messages of std::string type only. However, there are a few important points to mention here:

- This approach makes library and interfaces design much cleaner.
- HieLog provides simple (yet extensible) syntax for building strings.
- Logging functionality is totally separated from log_scope and log_message classes. Both tools are simple lightweight helpers and each might be replaced with custom tool providing more complex behavior.

An example of building string, containing arguments of various types is given below:

```
std::string message = hielog::string_formatter()
    << “Text ” << 0.123 << “ text ” << 124;
```

There is always a chance to extend it via overloading `operator <<` at global scope.

Concurrency

Key point of HieLog is correct implementation of concurrent logging. Most logging implement this feature either via locking log stream (via mutex or critical section) or transparently via using thread safe log target, which turns into the same case actually. This leads to a serious problem. Most logging libraries add additional (highly not desirable) synchronization to an application. This potentially leads to one or both consequences: application either works much slower or does not operate properly, when logging is turned off.

Due to the need of maintaining current state for each thread (indentation level), HieLog was supplied with internal storage, which is shared between threads. HieLog shares storage in the following way. Storage is represented via a map, where thread identifier acts as a key and thread specific data as a value. This map is synchronized by means of single writer/multiple readers pattern, where at least one writer prevent anyone to read controlling object and vice versa. So locking only occurs, when new thread is created, which is rather rear case. Each portion of thread specific data is dedicated to a thread and once it is extracted from the storage, there is no need to synchronize access to it.

```
typedef log<
    file_provider< multiple_files<> >,
    basic_formatter<> > log_type;
```

The log defined above points log streams coming from different threads to corresponding files. This happens absolutely transparent from user standpoint.

Networking

Streaming logs via TCP/IP is extremely useful, when target device file system is not easily accessible or device is tool small to fit detailed log. This is particularly topical for wireless devices. With all above mentioned limitations such devices are supplied with rich networking capabilities.

Networking in HieLog is implemented on top of Boost.Asio. This makes the solution even more stable and reliable. There is also an option to create connection per thread of execution. In this case the same concurrency backend as described above is involved.

```
typedef log<
    remote_provider,
    basic_formatter<> > log_type;
log_type log_instance("192.168.1.2", "1234");
```

Tutorial

Copyright

Cover Letter

Hello,

My name is Egor Pushkin. I'm mobile technology architect. I'm using Boost library for a couple of years and track its development status frequently.

What is really missing in Boost is logging. I think that everyone realizes that. I know that a number of libraries were reviewed, and one of them was even provisionally accepted. However, this area is so wide so it is hard to cover all use cases within the same tool and implement out of box solution, which meets all requirements.

I implemented library for hierarchical logging (HieLog) and would like to introduce it briefly:

- Adopts C++ approach to logging. It is written with all C++ aspects kept in mind. It is not overloaded with macros. It leverages advantages of other libraries from Boost package.

- Leverages advantages of correct concurrent design. It is designed to simplify debugging of complex applications with heavy concurrency utilization. Its implementation is intended to avoid additional synchronization overhead, when accessed from various threads.

- Implements generic, lightweight, extensible solution. It is exactly, what Boost is used to be. There are many examples, where people attempt to create complex, hard to understand and extend libraries for logging. No one such library corers everything. So the emphasis should be made on extensibility and design.

I attached detailed library overview and the source code.

All the attached materials are available at

http://hielog.googlecode.com/

Please, let me know whether such approach makes sense. Feel free to ask as many questions and make as many notes as necessary.

I'm particularly interested in passing through Boost submission process, so other developers have a chance to take advantage of this technology.

Best regards,

Egor Pushkin.