NAME

ErrLog – Shore Error Logging Class

SYNOPSIS

```
#include <syslog.h>

class ErrLog;

LogPriority {
    log_none = -1,            // none
    log_emerg = LOG_EMERG,    // no point in continuing (syslog's LOG_EMERG)
    log_fatal = LOG_ALERT,    // no point in continuing (syslog's LOG_ALERT)
    log_internal = LOG_CRIT,  // internal error
    log_error = LOG_ERR,      // client error
    log_warning = LOG_WARNING,// client warning
    log_info = LOG_INFO,      // just for yucks
    log_debug=LOG_DEBUG,      // for debugging gory details
};

// The following __omanip functions are defined to correspond
// to the LogPriority:
// emerg, fatal, internal, error, warning, info, debug
//
// Log messages must end with the new __omanip function
// flushl.
//
enum LoggingDestination {
    log_to_ether,       // no logging - for testing this package
    log_to_unix_file,
    log_to_open_file,
    log_to_syslogd,
    log_to_stderr
};

typedef void (*ErrLogFunc)(ErrLog *, void *);

class logstream; // forward
class ErrLog {
    ErrLog(
        const char *ident,
        LoggingDestination dest, // required
        void *arg = 0,          // one of :
                                // pathname,  (log_to_unixfile)
                                //   "-" means same as log_to_stderr
                                // FILE *,      (log_to_openfile)
                                // syslog facility (log_to_syslogd)
                                // ignored for log_to_stderr
        LogPriority level =  log_error,
        char *ownbuf = 0,
        int  ownbufsz = 0  // length of ownbuf, if ownbuf is given
    );
    ~ErrLog();

    // same name
```

```
    logstream   clog;
    void log(enum LogPriority prio, const char *format, ...);
    const char * ident();
    LoggingDestination  destination();
    LogPriority getloglevel();
    const char * getloglevelname();
    LogPriority setloglevel(LogPriority prio);
    static ErrLog *find(const char *id);
    static void apply(ErrLogFunc func, void *arg);
}
```

DESCRIPTION

The class `ErrLog` provides a unified, flexible interface to syslog and to Unix files for issuing errors or informational messages. A process can have many ErrLogs at once. Each ErrLog has an identity. If two ErrLogs are instantiated with the same `ident,` the class returns an error. The class keeps all instances of ErrLogs in a list so that an ErrLog can be located by its `ident` with the method **ErrLog::find.**

When an ErrLog is created, along with its identity, the caller must specify the ErrLog's destination, additional information that depends on the destination, the logging level for the log object, and the buffer to be used by the log for buffering messages. For example:

```
#include <errlog.h>

ErrLog *log_syslog =
   new ErrLog("syslog", log_to_syslogd, (void *)LOG_USER, log_error);
```

Log messages will be sent to the syslog daemon, under the facility name "user". Only messages of priority `log_error` and higher will be sent. The default buffer will be used to buffer the messages.

The class `ErrLog` has two mechanisms for generating log messages; one mechanism is compatible with C++ output streams; the other mechanism is similar to the `syslog()` function. To use C++ output-stream-style logging, in place of an `ostream` use the member of the `ErrLog` called `clog,` which is an object of class `logstream` (derived from `ostream`).

```
log_syslog->clog << warning << "Warning: Do not pass go." << flushl;
```

The message "Warning: Do not pass go" will not be sent because the warning priority is lower than the error priority. The log message ends with the __omanip function `flushl,` which causes the log to be flushed. The log is also flushed each time the priority of the message changes. For example, the following statement cause three distinct messages to be processed:

```
log_syslog->clog << debug << "testing" << error "Oops"
    << info << "interesting" << flushl;
```

Each of the messages has the priority given prior to the message, and the log is flushed four times: once before each priority change and at the end by `flushl.` The `__omanip` functions emerg, fatal, internal, error, warning, info, and debug are ignored if they are used with an `ostream` (such as cerr). The function `flushl` is equivalent to `flush` if it is used with an `ostream` rather than with a logstream.

In the following example, the caller gives a buffer for use by the log. (The default buffer is 1000 bytes in size; only if messages might exceed 1000 bytes in length is it necessary to provide a buffer.)

```
char   *big_buffer[10000];
ErrLog *log_stderr =
   new ErrLog("errors", log_to_stderr, (void *)0, log_error,
    big_buffer, sizeof(big_buffer));
log_stderr->log(log_error, "Error: %s passed go.", "George");
```

The message "Error: George passed go." will be printed on the standard error file because the priority of the message is the priority of the log. The method `log()` is used here to print the message; this style of logging can be mixed with the output-stream-style of logging shown in the previous example. Each invocation of `log()` flushes the log. This method is similar to the `syslog()` function call, but it does not recognize the "%m" format. The ErrLog class does not have the capability to handle "%m".

A log can be attached to an already-open file if the file has been opened with `fopen().` The constructor for ErrLog then takes a `FILE *` as its third parameter.

```
FILE     *f = fopen(...);
ErrLog *log_openfile =
    new ErrLog("information", log_to_open_file, f, log_info);
```

Finally, an ErrLog can be attached to a Unix file. The file is created if it does not exist, and if the file already exists, it is opened for appending.

```
ErrLog *log_file =
    new ErrLog("tracing", log_to_unix_file, "/my/path/debug.out", log_debug);
```

The ErrLog class does not log to Shore objects because it is meant not to rely on the Shore Value-Added Server being part of the process in which it is linked.

VERSION

This manual page applies to Version 1.1 of the Shore software.

SPONSORSHIP

The Shore project is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-91-C-Q518.