

Imagen

for Windows/Linux

OAL Module Documentation

Operating System Abstraction Layer

January 11, 2001. Document Revision 1

Netclime Inc.

P.O.Box 251666, LA, CA-90025
Tel (310) 571-3135 Fax (646) 514-0412

email: imagen@netclime.com

Responsibilities

Purpose

The purpose is to provide generic solutions to many common problems. It also encapsulates all 3rd party functions that are provided in OS specific, language specific or machine specific way. Syntax enhancements to make development more effective are included as well. The main goals are:

- Make code platform independent
- Code reuse
- Enhance syntax

Provided Functionality

- Timers
- Mutex
- Binary and text output
- Client TCP blocking connections
- Client SMTP connections for mail sending
- Wrappers of OS specific functions
- Fatal conditions exception handling

Interface

Integration Instructions

Required Headers and Header Paths

- oal.h
- output.h
- output_i.h
- timer.h
- mutex.h
- exception.h
- blocking_tcp.h
- smtp_connection.h

Required LIBs and LIB paths

Dependencies

C++ Objects

Namespaces

All functions and classes are placed in **namespace oal**, only network functions and classes reside in **namespace oal::net_client**.

Macros

Templates

Types (typedefs, structures, etc.)

struct oal::timer – an abstract class implementing a timer. Inherit this class and provide your own `virtual void time_expired()` member function, which will be called upon timeout expiration.

Public members:

timer(oal::ms delay) – constructor, initializing timer object; the time is specified in milliseconds and the measurement is started on timer construction

virtual ~timer() – virtual destructor, you can provide your own clean up procedure

virtual void time_expired() – override this function, providing your own behavior; the function will be called upon timeout expiration

OS specific:

- **Windows:** There **can exist** multiple timer derivatives in a process. The timer is implemented using a NT multimedia timer. It creates additional thread in the system – timer thread. Upon timer expiration the user code is run in the timer thread, in parallel with the other process threads.
- **Linux:** There **can not exist** multiple timer derivatives in a process. The timer is implemented by installing custom signal SIGALRM handler.

struct oal::exit_timer: `timer` – timer derivative calling `exit`, with the specified return code, upon timer expiration

struct oal::mutex - mutex synchronization object

Public members:

inline mutex(state initial=SIGNALLED,const char *name=0) – constructor creating the mutex object and setting its initial state and name

inline ~mutex() – destructor, freeing the resource

inline result wait(timeout timeout=INFINITE) const - blocks the calling thread/process until timeout elapses or the object ownership is obtained

inline result signal() const - releases the owned object

inline bool lock(timeout timeout=INFINITE) const – another way to say `wait()`

inline void unlock() const – another way to say `signal()`

OS specific:

- **Windows:** wrapper of Windows mutex synchronization object

- **Linux:** implemented on linux/unix semaphores; an array of two semaphores is created; the first semaphore is the mutex state, it can be 0 or 1; the second semaphore is a reference counter for mutex users, when this reference count becomes 0, meaning that no one is using the mutex anymore, the mutex (array of two semaphores) is unloaded from kernel;

struct oal::net_client::tcp_blocking_connection – class implementing client TCP blocking connections

Public members:

tcp_blocking_connection() – object constructor, allocates the socket resource

virtual ~tcp_blocking_connection() – object virtual destructor, if the connections is still opened; closes it (disconnects it); frees resources;

inline int connect(unsigned remote_host, unsigned short remote_port) – connects (opens connection) to a remote host on the specified port; both the ip address and the port are expected to be in i80x86 byte order (Least Significant Byte first). Return 0 if successful, otherwise error code;

inline int disconnect() – closes open connection; Return 0 if successful, otherwise error code;

inline int send(const char *what, int len) – sends data on open connection; Return 0 if successful, otherwise error code;

inline int receive(char *where, int len) – receives data on open connection; Return 0 if successful, otherwise error code;

Protected members:

inline int last_error() const – return the last error occurred

struct smtp_connection: private tcp_blocking_connection – client SMTP connection. Provides simple interface for mail sending.

```
bool send_mail(const char *smtp_server, // mail server name
              const char *from_name, // name of e-mail sender
              const char *from_email, // sender's e-mail
              const char *to, // recipient's e-mail
              const char *subject, // e-mail subject
              const char *body) // e-mail body
```

- sends an e-mail; On success returns true, otherwise false.

inline const std::string &describe_error() const – if send_mail() fails call this function in order to receive human readable error description.

Constants

Functions

void oal::init_exceptions() – initialization of fatal conditions handling

OS specific:

- **Windows:** The function creates static lifetime object which installs unhandled exceptions filter, and when an unhandled exception occurs the filter is called; it gathers information about the unhandled exception before the process exits.
- **Linux:** The function create numerous static lifetime objects which install signal handlers for appropriate signals. When a fatal condition occurs; the signal is received describing the type of fault.

inline unsigned oal::net_client::host(const char *host_name) – returns ip address from a host name; the ip address is in network byte order.

On the i80x86 the host byte order is Least Significant Byte first, whereas the network byte order, as used on the Internet, is Most Significant Byte first.

oal::ms oal::get_milliseconds() - returns milliseconds passed since some fixed time in the past modulus $2^{\text{bits-in-unsigned}}$, useful for calculating time deltas.

Note: Some implementations are inaccurate if used to measure intervals longer than 1 day.

void setup_process_dir(const char *argv0) -

the directory where the process is started from

std::string process_dir() -

std::string process_dir_slashed() -

Static Objects

Syntax Enhancements

Examples of Usage

Common Usage

Show here the simplest and most obvious method of use of the module. Include essential source fragments in the text. Put complete example in the appendix.

Tricky Usage

Show here the trickiest and not obvious method of use of the module. Include essential source fragments in the text. Put complete example in the appendix.

Design

Concept

Explain here all abstractions in the form of ideas methods and algorithms on which the module operation is based.

Ideas

Methods

Algorithms

Technologies

Structure

Overview of Internal Modules and Their Responsibilities

Explain functionality breakdown into modules. State each module's responsibilities.

Diagram of Modules Relationship (provide-use functionality)

In the terms of provide and use functionality show the relationship between modules

Data Flow and Control Flow Diagram

State here the stages through which the data processing goes. Show dispatching of control between objects/functions.