Nova library documentation

Version 1.0

Nova is a tiny C++ wrapper library over a larger set of libraries that provide OS functionalities over a variety of platform, for example, Windows and Linux. Nova has been written by Paul Fitzpatrick and documented by Giorgio Metta. Nova has been created as course/teaching support material about the basic concepts of operating systems programming.

Nova is distributed with a MS Visual Studio 6.0 SP6 project and Linux Makefile's. Nova is also distributed precompiled for MSVC 6.0 SP6 and Linux Debian gcc 3.xx.

One day this simple manual might evolve into proper Doxygen documentation.

Please send bug reports to Paul Fitzpatrick: paulfitz@csail.mit.edu
Please send amendments to this manual to Giorgio Metta: pasa@liralab.it

```
class NovaInit {
public:
  static void init();
  static void fini();
};
```

This class provides initialization and finalization functionalities to the library by constructing and destroy library-specific global objects.

The method *init()* must be called before any other class and/or library function is used. The method *fini()* is called before completion to free any memory allocated by the library.

```
class NovaThread {

public:
  NovaThread();
  virtual ~NovaThread();

  virtual void main();

  void begin();
  void end(double grace_period = -1);
  bool isEnding();

private:
  void *system_resource;

};
```

The *NovaThread* class provides encapsulation to the OS thread management routines. This class is designed to be used as a base class. The method *main()* can be overridden and represents the thread routine (where thread execution starts). The thread is started by the invocation of *begin()* and its termination is requested by calling *end()*. The thread cannot be signalled and thus it must be periodically checking the termination condition by calling *isEnding()*.

The grace_period variable of end() has the following meaning:

- -1: *end()* will wait (possibly forever) for the thread to actually terminate
- 0: the thread is terminated immediately
- >0: the thread is terminated after waiting for *grace_period* seconds

No error check is provided (for now) since thread creation should not fail in the simple context of this library.

```
class NovaTime {
public:
  static double now();
  static void sleep(double seconds);
};
```

NovaTime provides an interface to OS time-related functions. It contains two methods: *now()* which returns the current time as a *double* which represents the number of seconds since some suitable origin (e.g. Jan 1st, 1970), and *sleep()* which blocks the calling thread for the specified number of *seconds*.

```
class NovaSemaphore {

public:
  NovaSemaphore(int initial_count = 0);
  virtual ~NovaSemaphore();

  void wait();
  void post();
  bool check();

private:
  void *system_resource;

};
```

NovaSemaphore encapsulates the OS semaphore data structure. The constructor is used to assign the initial value to the semaphore count. The argument must be greater than 0. The method *wait()* is equivalent to the down operation, that is, if the count is 0 the calling thread goes to sleep otherwise the counter is decremented and the caller continues execution. Vice versa, the method *post()* increments the count by one and wakes up a sleeping thread on the same semaphore if any. *post*() will never block.

Finally the method *check()* verifies whether the caller would block, that is whether the semaphore count is equal to 0. It returns *true* iff at the time of the call the semaphore count is equal 0, otherwise it will decrement the count by one and behave as a normal *wait()*.

```
class NovaServer {
public:
  NovaServer();
  virtual ~NovaServer();

  int begin(int port);

  void accept(NovaClient& client);

private:
  void *system_resource;
};
```

The *NovaServer* represents a server side socket factory. Once started, it will wait for incoming connections. Upon reception, it allows accepting the connection and the instantiation of the appropriate socket to manage the communication channel.

The method to start listening for incoming connection is called *begin()*. The argument *port* is the TCP protocol port number (e.g. 9999, 8080). This method returns zero if initialization of the socket went ok.

To start accepting connections the user must call *accept()*. The call to *accept()* will block until a new valid connection is established. When *accept()* returns, the argument *client* is a new object of type *NovaClient* representing a communication channel which can be read or written.

```
class NovaClient {
public:

  NovaClient();
  virtual ~NovaClient();

  void connect(const char *hostname, int port);

  int send(const char *data, int len);
  int receive(char *data, int len, double timeout = -1);

private:
  void *system_resource;

  friend class NovaServer;
};
```

NovaClient encapsulates the communication socket of a TCP connection. It can act both on the client and server side. On the client side:

- The socket is created (constructor).
- The channel is connected by calling *connect(hostname, port)* where the *hostname* can be replaced also with the remote IP address and the *port* argument is the TCP port number.
- Aftter a successful connection either *send()* or *receive()* calls can be issued.

On the server side a *NovaClient* is returned by calling the *accept()* method of a *NovaServer* object. In this case the *NovaClient* object is already connected to the remote peer. The *send()* and *receive()* methods are available as before.

The *send()* method requires a pointer to the buffer with the *data* to send and the *length* of this buffer. The receive method requires the pointer to the *data* buffer, its size (*len*), and a *timeout* period in seconds.

receive() returns the number of bytes received or a negative number in case of failure/error. 0 is returned to indicate the EOF.

send() returns the number of bytes transferred or a negative number in case of failure/error.