JThread manual (v1.0.0)

Jori Liesenborgs
jori@lumumba.luc.ac.be

February 12, 2001

1 Introduction

A lot of projects on which I'm working use threads. To be able to use the same code on both unix and MS-Windows platforms, I decided to write some simple wrapper classes for the existing thread functions on those platforms.

The JThread package is very simple: currently, it only contains two classes, namely `JThread` and `JMutex`. As their names suggest, `JThread` represents a thread and `JMutex` a mutex. The thread class only contains very basic functions, for example to start or kill a thread.

2 Copyright & disclaimer

3 Usage

Here follows a description of the `JThread` and `JMutex` classes. Note that functions with return type `int` always return a value of zero or more on success and a negative value in case something went wrong.

3.1 JMutex

The class definition of `JMutex` is shown below. Before you can use an instance of this type, you must first call the `Init` function. You can check if the mutex was already initialized by checking the return value of `IsInitialized`. After the initialization, the mutex can be locked and unlocked by calling the functions `Lock` and `Unlock` respectively.

```
class JMutex
{
public:
    JMutex();
    ~JMutex();
    int Init();
    int Lock();
    int Unlock();
    bool IsInitialized();
};
```

3.2 JThread

To create your own thread, you have to derive a class from `JThread`, which is depicted below. In your derived class, you have to implement a member function `Thread`, which will be executed in the new thread.

To start your thread, you simply have to call the `Start` function. You can check if the thread is still running by calling `IsRunning`. If the thread has finished, you can check its return value by calling `GetReturnValue`. Finally, in case your thread gets stuck, you can end it by using the `Kill` function.

You should be careful with this `Kill` function: if you call it when the thread is working with a mutex (for example an internal mutex), this mutex can be left in a locked state, which in turn can cause another thread to block. You should only use the `Kill` function when you're absolutely sure that the thread is stuck in some loop and cannot be ended otherwise.

```
class JThread
{
public:
    JThread();
    virtual ~JThread();
    int Start();
    int Kill();
    virtual void *Thread() = 0;
    bool IsRunning();
    void *GetReturnValue();
};
```