Subject: sMutexLock implementation
Posted by hojtsy on Thu, 22 May 2008 08:38:33 GMT
View Forum Message <> Reply to Message

In Mt.cpp there is

```
Mutex& sMutexLock()
{
 static Mutex *section;
 if(!section) {
  static byte b[sizeof(Mutex)];
  section = new(b) Mutex;
 }
 return *section;
}
```

How is this different from the simpler

```
Mutex& sMutexLock()
{
 static Mutex m;
 return m;
}
```

In both cases the Mutex constructor will be called when the function is first called. In both cases the function needs external protection from MT race conditions.

On a side note, this function is not on the interface (Mt.h), why not make it file static in Mt.cpp to avoid name clashes?

Subject: Re: sMutexLock implementation
Posted by mirek on Fri, 30 May 2008 17:11:41 GMT
View Forum Message <> Reply to Message

hojtsy wrote on Thu, 22 May 2008 04:38In Mt.cpp there is

```
Mutex& sMutexLock()
{
 static Mutex *section;
 if(!section) {
  static byte b[sizeof(Mutex)];
  section = new(b) Mutex;
 }
 return *section;
}
```

How is this different from the simpler

```
Mutex& sMutexLock()
{
```

```
 static Mutex m;
 return m;
}
```
In both cases the Mutex constructor will be called when the function is first called. In both cases the function needs external protection from MT race conditions.

On a side note, this function is not on the interface (Mt.h), why not make it file static in Mt.cpp to avoid name clashes?

I think you are basically right. This was attempt to make more sure that initialization really happens and there is not catch, because external protection is not possible there.... as this is a mutex used to protect synchronization of other mutexes..

So the whole thing is meant to be run before second thread starts (and it is called in several places to ensure this).

I guess it is still better this way, as you never know what really happens in "static magic".

I agree about "static" though.

Mirek

Mirek
