
Subject: Re: DnD hangs in MT Refresh()ing
Posted by [kohait00](#) on Fri, 29 Jan 2010 09:31:09 GMT
[View Forum Message](#) <> [Reply to Message](#)

one more:

```
indeed, in CtrlMT.cpp  
void Ctrl::ICall(Callback cb)
```

is suffering a not release of semaphore due to an early CtrlCall.Clear in a DoCall somewhere in between, maybe from another thread??, so that the PerformCall cant be executed, which would release the semaphore...is there any other thread accessing DoCall?

a short log, sem1 is the ones from ICall, simpli with an id counter, sem2 is the ones from Call interfaces.

..

```
DoCall  
Sem release, sem1 934 sem2 0  
DoCall::CtrlCall clear sem1 934 sem2 0  
SEM1 released 934  
SEM1 wait 935  
DoCall  
Sem release, sem1 935 sem2 0  
DoCall::CtrlCall clear sem1 935 sem2 0  
SEM1 released 935  
SEM1 wait 936  
DoCall  
Sem release, sem1 936 sem2 0  
SEM1 released 936  
DoCall::CtrlCall clear sem1 936 sem2 0  
SEM1 wait 937  
DoCall  
DoCall::CtrlCall clear sem1 937 sem2 0  
DoCall  
DoCall::CtrlCall clear sem1 937 sem2 0  
DoCall  
DoCall::CtrlCall clear sem1 937 sem2 0
```

the CtrlMt.cpp used for that is the current one, from yesterday svn, so you can see where what is logged

```
#include "CtrlCore.h"
```

```
#define LLOG(x) // LOG(x)
```

NAMESPACE_UPP

```
#ifdef _MULTITHREADED
```

```
static int sem1 = 0;
```

```
static int sem2 = 0;
```

```
static int      NonMain;
```

```
static StaticMutex NonMainLock;
```

```
void EnterGuiMutex()
```

```
{
    Mutex & m = NonMainLock.Get();
    LLOG("Thread " << IsMainThread() << " trying to lock");
    bool nonmain = !IsMainThread();
    if(nonmain)
        NonMainLock.Enter();
    EnterGMutex();
    if(nonmain)
        NonMain++;
    LLOG("Thread " << IsMainThread() << " LOCK");
}
```

```
void LeaveGuiMutex()
```

```
{
    Mutex & m = NonMainLock.Get();
    LLOG("Thread " << IsMainThread() << " trying to unlock");
    bool nonmain = !IsMainThread();
    if(nonmain)
        NonMain--;
    LeaveGMutex();
    if(nonmain)
        NonMainLock.Leave();
    LLOG("Thread " << IsMainThread() << " UNLOCK");
}
```

```
struct Ctrl::CallBox {
```

```
    Semaphore sem;
```

```
    Callback cb;
```

```
};
```

```
void Ctrl::PerformCall(Ctrl::CallBox *cbox)
```

```
{
    cbox->cb();
    LOG("Sem release, sem1 " << sem1 << " sem2 " << sem2);
    cbox->sem.Release();
}
```

```
Callback  Ctrl::CtrlCall;
```

```

void WakeUpGuiThread();

bool Ctrl::DoCall()
{
    LOG("DoCall");
    CtrlCall();
    LOG("DoCall::CtrlCall clear sem1 " << sem1 << " sem2 " << sem2);
    CtrlCall.Clear();
    LLOG("--- DoCall, nonmain: " << NonMain);
    int a = NonMain;
    return NonMain;
}

void Ctrl::ICall(Callback cb)
{
    LLOG("Ctrl::Call " << IsMainThread() << ", nonmain: " << NonMain);
    if(IsMainThread())
        cb();
    else {
        CallBox cbox;
        cbox.cb = cb; ++sem1;
        CtrlCall = callback1(PerformCall, &cbox);
        int level = LeaveGMutexAll();
        WakeUpGuiThread();
        LLOG("Waiting for semaphore");
        if(!Thread::IsShutdownThreads())
        {
            LOG("SEM1 wait " << sem1);
            cbox.sem.Wait();
            LOG("SEM1 released " << sem1);
        }
        EnterGMutex(level);
    }
    LLOG("-- Ctrl::Call " << IsMainThread());
}

void Ctrl::Call(Callback cb)
{
    if(IsMainThread())
        cb();
    else {
        CallBox cbox;
        cbox.cb = cb; ++sem2;
        UPP::PostCallback(callback1(PerformCall, &cbox));
        int n = NonMain;
        int nn = n;
        NonMain = 0;
    }
}

```

```

for(int i = 0; i < n; i++)
    NonMainLock.Leave();
int level = LeaveGMutexAll();
WakeUpGuiThread();
if(!Thread::IsShutdownThreads())
{
    LOG("SEM2 wait " << sem2);
    cbox.sem.Wait();
    LOG("SEM2 released " << sem2);
}
for(int i = 0; i < n; i++)
    NonMainLock.Enter();
EnterGMutex(level);
NonMain = n;
}
}

#else

bool Ctrl::DoCall()
{
    return false;
}

void Ctrl::ICall(Callback cb)
{
    cb();
}

void Ctrl::Call(Callback cb)
{
    cb();
}
#endif

void Ctrl::GuiSleep(int ms)
{
    Call(callback1(&Ctrl::GuiSleep0, ms));
}

void Ctrl::WndDestroy()
{
    ICall(callback(this, &Ctrl::WndDestroy0));
}

#ifdef PLATFORM_WIN32
void Ctrl::WndCreateCaret(const Rect& cr)
{

```

```

    ICall(THISBACK1(WndCreateCaret0, cr));
}
#endif

void Ctrl::WndShow(bool b)
{
    ICall(THISBACK1(WndShow0, b));
}

void Ctrl::WndUpdate()
{
    ICall(THISBACK(WndUpdate0));
}

void Ctrl::SetWndForeground()
{
    ICall(THISBACK(SetWndForeground0));
}

bool Ctrl::WndEnable(bool b)
{
    ICall(THISBACK1(WndEnable0, &b));
    return b;
}

bool Ctrl::SetWndFocus()
{
    bool b;
    ICall(THISBACK1(SetWndFocus0, &b));
    return b;
}

void Ctrl::WndInvalidateRect(const Rect& r)
{
    ICall(THISBACK1(WndInvalidateRect0, r));
}

void Ctrl::WndSetPos(const Rect& rect)
{
    ICall(THISBACK1(WndSetPos0, rect));
}

void Ctrl::WndUpdate(const Rect& r)
{
    ICall(THISBACK1(WndUpdate0r, r));
}

void Ctrl::WndScrollView(const Rect& r, int dx, int dy)

```

```
{  
  ICall(THISBACK3(WndScrollView0, r, dx, dy));  
}
```

```
void Ctrl::EventLoop(Ctrl *ctrl)  
{  
  Call(callback1(&Ctrl::EventLoop0, ctrl));  
}
```

```
END_UPP_NAMESPACE
```
