Archived:How to decode images synchronously using Symbian C++

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Overview

The Symbian OS C++ image decoding API CImageDecoder is designed to work in an asynchronous manner enabled by AO (Active Object). However for some reasons (for example to simplify the program logic) developers may want to decode image synchronously. This article describes how to decode an image file or an image buffer in a real synchronous manner.

Solution

Decoding Image File

Many trials have proved that using User::WaitForRequest() to wait for the completion of CImageDecoder::Convert() will freeze the application, the rootcause is that by default the decoding runs in the calling thread via non-preemptive AO mechanism.

```
// Note: the code is NOT gonna work!
CImageDecoder* decoder = CImageDecoder::FileNewL(fs, fileName);
CleanupStack::PushL(decoder);
TFrameInfo frameInfo = decoder->FrameInfo(0);
CFbsBitmap *tex = new(ELeave) CFbsBitmap;
err = tex->Create(frameInfo.iOverallSizeInPixels, frameInfo.iFrameDisplayMode);
User::LeaveIfError(err);
CleanupStack::PushL(tex);
TRequestStatus status = KRequestPending;
decoder->Convert( &status, *tex);
User::WaitForRequest(status); // will wait forever...
CleanupStack::Pop(tex);
CleanupStack::PopAndDestroy(decoder);
```

The problem can kind of be resolved by Nested Active Scheduler, i.e. start a new event loop by CActiveSchedulerWait, but this way we can only implement "pseudo synchronous", means during decoding the application can still handle UI events or other events. It can make the program logic out of control if, for example, the UI events handling code try to delete the decoder in the middle of decoding processes.

To truly resolve the problem we should use EOptionAlwaysThread to instruct the decoder to decode the image file in a separate thread.

```
CFbsBitmap* MyPicLoader::LoadL(const TDesC& aFileName)
    {
    RFs fs;
    TInt err = fs.Connect();
    User::LeaveIfError(err);
    CleanupClosePushL(fs);

    CImageDecoder* decoder = CImageDecoder::FileNewL(fs, aFileName, CImageDecoder::EOptionAlwaysThread);
    CleanupStack::PushL(decoder);
    TFrameInfo frameInfo = decoder->FrameInfo(0);
    CFbsBitmap *tex = new(ELeave) CFbsBitmap;
    err = tex->Create(frameInfo.iOverallSizeInPixels, frameInfo.iFrameDisplayMode);
    User::LeaveIfError(err);
    CleanupStack::PushL(tex);
    TRequestStatus status = KRequestPending;
    decoder->Convert( &status, *tex);
    User::WaitForRequest(status);            // Convert() is asynchronous, wait on it
    CleanupStack::Pop(tex);
    CleanupStack::PopAndDestroy(decoder);

    CleanupStack::PopAndDestroy(&fs);

    return tex;
    }
```

Decoding Image Buffer

Regarding decoding image buffer, a new problem is that the EOptionAlwaysThread seems not implemented for it at all. Even if we add EOptionAlwaysThread to DataNewL the User::WaitForRequest() will still wait forever.

```
// Note: EOptionAlwaysThread DOESN'T work for CImageDecoder::DataNewL
CFbsBitmap* MyPicLoader::LoadL(const TDesC8* aBuffer)
    {
    RFs fs;
    TInt err = fs.Connect();
    User::LeaveIfError(err);
    CleanupClosePushL(fs);
```

http://www.developer.nokia.com/Community/Wiki/Archived:How_to_decode_images_synchronously_using_Symbian_C%2B%2B(C) Copyright Nokia 2013. All rights reserved.

```
    CImageDecoder* decoder = CImageDecoder::DataNewL(fs, *aBuffer, CImageDecoder::EOptionAlwaysThread);
    CleanupStack::PushL(decoder);
    TFrameInfo frameInfo = decoder->FrameInfo(0);
    CFbsBitmap *tex = new(ELeave) CFbsBitmap;
    err = tex->Create(frameInfo.iOverallSizeInPixels, frameInfo.iFrameDisplayMode);
    User::LeaveIfError(err);
    CleanupStack::PushL(tex);
    TRequestStatus status = KRequestPending;
    decoder->Convert( &status, *tex);
    User::WaitForRequest(status); // will wait forever...
    CleanupStack::Pop(tex);
    CleanupStack::PopAndDestroy(decoder);

    CleanupStack::PopAndDestroy(&fs);

    return tex;
    }
```

Of course we can always solve the issue by Nested Active Scheduler, but to implement real synchronous we have to create our own thread for decoding.

```
struct ThreadParameters
    {
    const TDesC8* iBuffer; // input
    TInt iHandle; // handle of the decoded bitmap object
    };

void ThreadMainL(ThreadParameters* aParam)
    {
    CMyImageLoader* loader = CMyImageLoader::NewLC();
    loader->LoadL(aParam->iBuffer, aParam->iHandle); // send the decoding request
    CActiveScheduler::Start(); // and then start the message loop
    User::LeaveIfError(loader->iStatus.Int());
    CleanupStack::PopAndDestroy(loader);
    }

TInt ThreadFunc(TAny* aParam)
    {
    TInt err = KErrNone;
    ThreadParameters* param = reinterpret_cast<ThreadParameters*>(aParam);
    CTrapCleanup *cs = CTrapCleanup::New(); // create infrastructures like cleanup stack and active scheduler for the thread
    if (cs)
        {
        CActiveScheduler *s = new CActiveScheduler;
        if(s)
            {
            CActiveScheduler::Install(s);
            TRAP(err, ThreadMainL(param));
            delete s;
            }
        delete cs;
        }
    else
        {
        err = KErrNoMemory;
        }
    return err;
    }


// thread version
CFbsBitmap* MyPicLoader::LoadL(const TDesC8* aBuffer)
    {
    RFs fs;
    TInt err = fs.Connect();
    User::LeaveIfError(err);
    CleanupClosePushL(fs);
    CImageDecoder* decoder = CImageDecoder::DataNewL(fs, *aBuffer);
    TFrameInfo frameInfo = decoder->FrameInfo(0);
    delete decoder;
    CleanupStack::PopAndDestroy(&fs);

    CFbsBitmap* bitmap = new(ELeave) CFbsBitmap;
    CleanupStack::PushL(bitmap);
    err = bitmap->Create(frameInfo.iOverallSizeInPixels, frameInfo.iFrameDisplayMode);
    User::LeaveIfError(err);
    ThreadParameters param;
    param.iBuffer = aBuffer;
    param.iHandle = bitmap->Handle();
    RThread thread;
    _LIT(KThreadName, "thread1" );
    thread.Create(KThreadName, ThreadFunc, KDefaultStackSize, NULL, &param); // create a new thread
    User::LeaveIfError(err);
    CleanupClosePushL(thread);
    thread.Resume(); // then let the thread do the job
    TRequestStatus status = KRequestPending;
    thread.Logon(status);
    User::WaitForRequest(status); // and then wait until the thread exits
    User::LeaveIfError(status.Int());
    CleanupStack::PopAndDestroy(&thread);
    CleanupStack::Pop(bitmap);
    return bitmap;
    }
```

More details about how to use the image decoder in a new thread please refer to the full example.

Note that REComSession::FinalClose() shall be called before you end the thread to avoid memory leak.

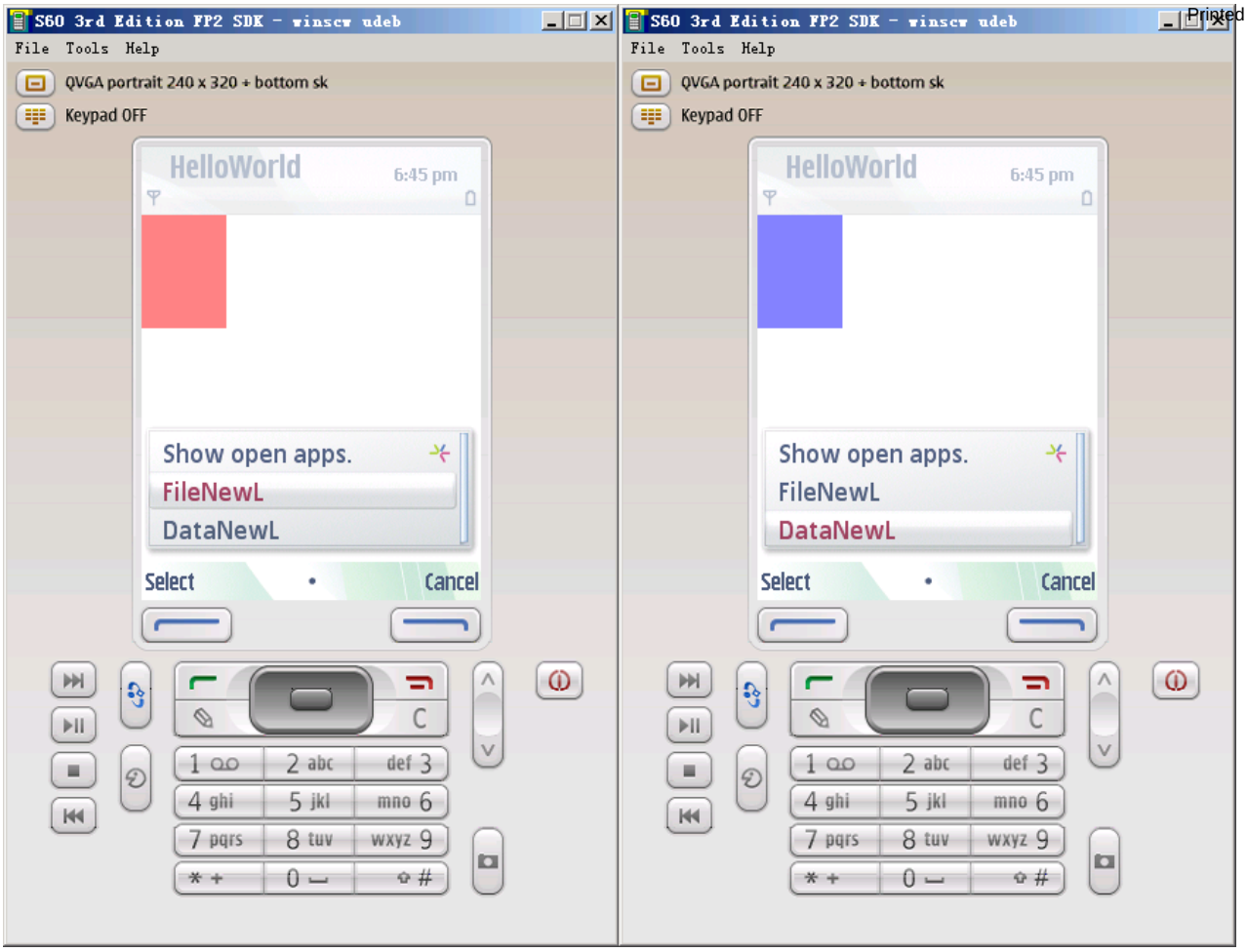
Source Code

Full example (you may need to change the hard-coded png path in source code in order to run it on target):

HelloWorld(SynchronousImageBufferDecoder).zip

Screenshot (first: decoding image file, second: decoding image buffer):

http://www.developer.nokia.com/Community/Wiki/Archived:How_to_decode_images_synchronously_using_Symbian_C%2B%2B(C) Copyright Nokia 2013. All rights reserved.



Relative documents

http://www.developer.nokia.com/Community/Wiki/Archived:How_to_decode_images_synchronously_using_Symbian_C%2B%2B(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.