Archived:Changing asynchronous method to synchronous in 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

This code snippet shows how to change an asynchronous method interface to a synchronous method. This can be done with CActiveSchedulerWait.

There are two different synchronous methods:

- CMyClass::DoAsynchronousL() uses CActiveSchedulerWait for waiting for the asynchronous method that completes with CMyClass::AsyncComplete()
- CMyClass::DoAsynchronousWithStatusL() uses CActiveSchedulerWait for waiting for the asynchronous method that has TRequestStatus in its interface. The method completes with CMyClass::RunL().

For more information on using CActiveSchedulerWait, see article Archived:Using CActiveSchedulerWait

MMP file

The following capabilities and libraries are required:

```
CAPABILITY None
LIBRARY euser.lib
```

Header

```
#include <e32base.h>

// for CSomeAsyncClass
class MAsynchIF
    {
    public:
        void AsyncComplete(TInt aResult);
    };

class CMyClass : public CActive, MAsynchIF
     {
     public:
         static CMyClass* NewLC();
         virtual ~CMyClass();

         // Synchronous methods
         TInt DoAsynchronousL();
         TInt DoAsynchronousWithStatusL();

     private: // From MAsynchIF (CSomeAsyncClass)
         void AsyncComplete(TInt aResult);

     private: // From CActive
         void RunL();
         void DoCancel();

     private:
         CSomeAsyncClass*       iSomeAsyncClass;
         CActiveSchedulerWait iSchedulerWait;
         TInt iResult;
    };
```

Source

```
CMyClass* CMyClass::NewLC()
    {
    CMyClass* self = new (ELeave) CMyClass();
    CleanupStack::PushL(self);
    self->ConstructL();
    return self;
    }

CMyClass::CMyClass()
: CActive(CActive::EPriorityStandard)
    {
    }

void CMyClass::ConstructL()
    {
    CActiveScheduler::Add(this);
    iSomeAsyncClass = CSomeAsyncClass::NewL(this);
    }

CMyClass::~CMyClass()
    {
    Cancel();
    delete iSomeAsyncClass;
```


© Copyright Nokia 2013. All rights reserved.

```
    }

// This is a synchronous method even though it calls an asynchronous interface that
// completes with CMyClass::AsyncComplete()
TInt CMyClass::DoAsynchronousL()
    {
    User::LeaveIfError(iSomeAsyncClass->SomeAsynchronousIfL());

    iSchedulerWait.Start();
    // Continues here after CMyClass::AsyncComplete() is called
    // by CSomeAsyncClass.

    return iResult;
    }

void CMyClass::AsyncComplete(TInt aResult)
    {
    iResult = aResult;
    iSchedulerWait.AsyncStop();
    }

// This is an synchronous method even though it calls an asynchronous interface that
// completes with CMyClass::RunL() because it uses TRequestStatus as
// a parameter
TInt CMyClass::DoAsynchronousWithStatusL()
    {
    if (!IsActive())
        {
        User::LeaveIfError(iSomeAsyncClass->SomeAsynchronousIfL(iStatus));
        SetActive();

        iSchedulerWait.Start();
        // Continues here after CMyClass::RunL() is called
        // by CSomeAsyncClass.
        }
    else
        {
        User::Leave(KErrNotReady);
        }
    return iResult;
    }

void CMyClass::RunL()
    {
    // This is called when DoAsynchronousWithStatusL() completes
    iResult = iStatus.Int();
    iSchedulerWait.AsyncStop();
    }

void CMyClass::DoCancel()
    {
    // This cancels the DoAsynchronousWithStatusL() method
    iResult = KErrCancel;
    iSchedulerWait.AsyncStop();
    }
```

Postconditions

The asynchronous methods looks like a synchronous method.

See also

Archived:Using CActiveSchedulerWait

© Copyright Nokia 2013. All rights reserved.

© Copyright Nokia 2013. All rights reserved.