Archived:Using CActiveSchedulerWait

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

CActiveSchedulerWait controls a single scheduling loop in the current global CActiveScheduler active scheduler. CActiveSchedulerWait provides better control of nested wait loops in the active scheduler.

Note that a CActiveSchedulerWait object can be used as a data member inside other CBase-derived classes.

See also the code example about threads Archived:RThread - Symbian threading example.

Source: creating and starting CActiveSchedulerWait

Execute RThread:

```
TInt CSomeThread::ExecuteThread(TAny *aPtr)
    {
    // Create cleanupstack
    CTrapCleanup* cleanupStack = CTrapCleanup::New();

    // Create active sheduler and start thread
    TRAPD( error, CThreadAOEngine::CreateActiveSchedulerL( sharedMediator ) )

    delete cleanupStack;
    return 0;
    }
```

Start the active scheduler and the asynchronous operation:

```
void CSomeThread::CreateActiveSchedulerL()
    {
    // 1. Create a new active scheduler.
    CActiveScheduler* activeScheduler = new (ELeave) CActiveScheduler;
    CleanupStack::PushL(activeScheduler);

    // 2. Use the static function Install to install the previously created scheduler.
    CActiveScheduler::Install(activeScheduler);

    // 3. Created a nested loop inside CActiveScheduler.
    CActiveSchedulerWait* wait = new (ELeave) CActiveSchedulerWait;
    CleanupStack::PushL(wait);

    // 4. Create a class that must be handled asynchronously.
    CSomeAsyncClassTimer* timer = CSomeAsyncClassTimer::NewL(wait);
    CleanupStack::PushL(timer);

    // 5. Add an active object in CActiveScheduler.
    CActiveScheduler::Add(timer);

    // 6. Active scheduler must have one outstanding request before it can
    //    be started.
    timer->StartL();

    // 7. Start the nested scheduler loop.
    wait->Start();


    // 8. The process continues here after calling CActiveSchedulerWait::AsyncStop()
    //    that is called in CSomeAsyncClassTimer::StopWaitLoop()


    // Remove and delete the scheduler and the rest.
    CleanupStack::PopAndDestroy(3);
    }
```

Source: stopping CActiveSchedulerWait

```
void CSomeAsyncClassTimer::StartL()
    {
    // TODO: An asynchronous operation starts in
    // this CActive object
    }

void CSomeAsyncClassTimer::RunL()
    {
    // Asynchronous operation stops

    // We desided to stop our own CActiveSchedulerWait
    StopWaitLoop();
```


(C) Copyright Nokia 2013. All rights reserved.

```
    // NOTE: This does not stop the global CActiveScheduler but it continues
    // running.
    }

void CSomeAsyncClassTimer::StopWaitLoop()
    {
    Cancel();
    // Stops our own nested scheduler loop
    iActiveSchedulerWait->AsyncStop();
    }
```

Postconditions

CActiveSchedulerWait, a nested wait loop, has been created in global CActiveScheduler. CActiveSchedulerWait stopped after running the asynchronoys operation. CActiveSchedulerWait does not stop the global CActiveScheduler but it continues running.

See also

Archived:RThread - Symbian threading example

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.