Logs monitoring Example

The `CLogMonitor` illustrates how to monitor log entries as they are written to the logs by the OS. Log events are reported using the `LogEventL()` callback method. Note that each call might contain multiple log events. First might be “missed” and some time later it could be changed to “answered”. If you are deleting entries, make sure you delete them all.

If you want to read all entries currently stored in the logs you can use CCallLogReader instead.

Also an example for monitoring and deleting log entries for SMS messages, especially for Delivery reports is also illustrated in SMS DeliveryReport Deleting Example.

Link against:

```
LIBRARY logcli.lib
```

Capabilities:

```
CAPABILITY ReadUserData
```

Log_Monitor.h

```
#include <logcli.h>
#include <logview.h>

class MLogMonitor
{
    public:
        virtual void LogEventL(const CLogEvent& event) = 0;
};

class CLogMonitor : public CActive
{
    enum TMonitorStates
    {
        EUninitalised,
        EInitalised,
        EWaitingChange,
        EReadingLog,
        EReadingFirstLog,
        EDeletingEvent,
        EReadingLogItems
    };

    public:
        static CLogMonitor* NewL(MLogMonitor* aCallBack);
        static CLogMonitor* NewLC(MLogMonitor* aCallBack);
        ~CLogMonitor();

    protected:
        CLogMonitor(MLogMonitor* aCallBack);
        void ConstructL(void);
        void StartMonitorL();
        void GetLatest();
        void GetFirstEventL();
        void GetNextEventL();
        void DoCancel();
        void RunL();

    private:
        CLogClient*     iLogClient;
        CLogViewRecent* iRecentLogView;
        TMonitorStates  iState;
        MLogMonitor*    iCallBack;
        RFs             iFsSession;
};
```

Log_Monitor.cpp

```
#include "Log_Monitor.h"

CLogMonitor* CLogMonitor::NewL(MLogMonitor* aCallBack)
{
    CLogMonitor* self = CLogMonitor::NewLC(aCallBack);
    CleanupStack::Pop(self);
    return self;
}

CLogMonitor* CLogMonitor::NewLC(MLogMonitor* aCallBack)
{
    CLogMonitor* self = new (ELeave) CLogMonitor(aCallBack);
    CleanupStack::PushL(self);
    self->ConstructL();
    return self;
}

CLogMonitor::CLogMonitor(MLogMonitor* aCallBack)
:CActive(0),iState(EUninitalised),iCallBack(aCallBack)
{
}
```


(C) Copyright Nokia 2013. All rights reserved.

```
CLogMonitor::~CLogMonitor()
{
    Cancel();

    delete iRecentLogView, iRecentLogView = NULL;
    delete iLogClient, iLogClient = NULL;

    iFsSession.Close();
}

void CLogMonitor::ConstructL()
{
    CActiveScheduler::Add(this);

    User::LeaveIfError(iFsSession.Connect());

    iLogClient = CLogClient::NewL(iFsSession);
    iRecentLogView = CLogViewRecent::NewL(*iLogClient);

    iState = EInitalised;
    StartMonitorL();
}

void CLogMonitor::DoCancel()
{
    if(iRecentLogView)
        iRecentLogView->Cancel();

    if(iLogClient)
    {
        if(iState == EWaitingChange)
            iLogClient->NotifyChangeCancel();
        else
            iLogClient->Cancel();
    }
}

void CLogMonitor::RunL()
{
    if(iStatus != KErrCancel)
        switch(iState)
        {
            case EWaitingChange:
                // if this doesn't appear to catch the event right
                // you could try sleeping a bit before fetching the event...
                // User::After(2000000);
                GetLatest();
                break;

            case EReadingLog:
                if(iRecentLogView)
                {
                    if(iRecentLogView->CountL() > 0)
                        GetFirstEventL();
                    else
                        StartMonitorL();
                }
                break;

            case EReadingFirstLog:
            case EReadingLogItems:
                if(iStatus == KErrNone && iRecentLogView)
                {
                    iCallBack->LogEventL(iRecentLogView->Event());
                    GetNextEventL();
                }
                else
                    StartMonitorL();
                break;

            case EDeletingEvent:
                GetNextEventL();
                break;

            default:
                StartMonitorL();
                break;
        }
}


void CLogMonitor::StartMonitorL()
{
    if(iLogClient)
    {
        if(iRecentLogView)
            iRecentLogView->Cancel();

        iLogClient->Cancel();

        iState = EWaitingChange;
        iLogClient->NotifyChange(TTimeIntervalMicroSeconds32(3000000),iStatus);
        SetActive();
    }
}

void CLogMonitor::GetLatest()
{
    if(iRecentLogView)
    {
        iState = EReadingLog;
        iRecentLogView->Cancel();
        if(iRecentLogView->SetRecentListL(KLogNullRecentList,iStatus))
            SetActive();
        else
            StartMonitorL();
    }
}

void CLogMonitor::GetFirstEventL()
{
    if(iRecentLogView)
    {
        iState = EReadingFirstLog;
        if(iRecentLogView->LastL(iStatus))
            SetActive();
        else
            StartMonitorL();
    }
```


(C) Copyright Nokia 2013. All rights reserved.

```
}


void CLogMonitor::GetNextEventL()
{
    if(iRecentLogView)
    {
        iState = EReadingLogItems;
        if(iRecentLogView->PreviousL(iStatus))
            SetActive();
        else
            StartMonitorL();
     }
}
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.