



This code is to monitor for file changes (added, erased) in a particular directory.

ID		Creation date	March 12, 2008
Platform	S60 3rd Edition S60 3rd Edition, FP1	Tested on devices	N73
Category	Symbian C++	Subcategory	File/Data

Keywords (APIs, classes, methods, functions):

Contents

- [1 Overview](#)
- [2 MMP file \(optional\)](#)
- [3 Libraries needed](#)
- [4 Header file](#)
- [5 Source file](#)
- [6 Postconditions](#)
- [7 Test application and other attachments](#)

Overview

This code shows how to monitor for file changes (added/erased) in a particular directory. There is also an application showing how to use the code [Source Code](#)

This snippet can be self-signed.

MMP file (optional)

The following capabilities and libraries are required:

```
CAPABILITY ReadUserData
```

Libraries needed

```
LIBRARY euser.lib //RTimer, CActive
```

Header file

```

/*
=====
Name          : DirectoryMonitor.h
Author       : Olympio Cipriano
Version      : 1.0
Copyright    :
Description  : CDirectoryMonitor declaration
=====
*/

#ifndef CDirectoryMonitor_H
#define CDirectoryMonitor_H

#include <e32base.h>    // For CActive, link against: euser.lib
#include <e32std.h>    // For RTimer, link against: euser.lib
#include <f32file.h>

enum TFileAction
{
    EFileErased,
    EFileAdded
};

class DirectoryObserver
{
public:
    virtual void DirectoryChanged( const TDesC& aFileName, TFileAction aAction ) = 0;
};

class CDirectoryMonitor : public CActive
{
public:
    // Cancel and destroy
    ~CDirectoryMonitor()

    // Two-phased constructor.
    static CDirectoryMonitor* NewL( DirectoryObserver * iObserver );

    // Two-phased constructor.
    static CDirectoryMonitor* NewLC( DirectoryObserver * iObserver );

public:
    // New functions
    // Function for making the initial request
    void Monitor( const TDesC& aDirPath );

private:
    // C++ constructor
    CDirectoryMonitor( DirectoryObserver* aObserver );

    // Second-phase constructor
    void ConstructL();

    //directory internals
    void ListFiles( CDir*& aDir );

    void DiffDirectoryContents( );

```

Directory_Monitoring

```
bool SearchEntry( const TEntry& aEntry, CDir* aDir );

void Clean();

private:
// From CActive
// Handle completion
void RunL();

// How to cancel me
void DoCancel();

// Override to handle leaves from RunL(). Default implementation causes
// the active scheduler to panic.
    TInt RunError();

struct TFileChange
{
    (const TFileChangeEntry, TFileAction aAction)
        :fileEntry(aEntry),action(aAction){}
const TEntry fileEntry;
        TFileAction action
};

private:
enum TCDirectoryMonitorState
{
    EUninitialized,
    //EInitialized,
    // Error condition
};

private:
    TInt iState of the active object
    ;RFS iFs
    DirectoryObserver;
    * iOCDDir;
    * iNCDDir;
    <TFileChange> iFileChangeArray;
    <TArray> iFileList;
    * iButton;
};

#endif // CDirectoryMonitor_H
```

Source file

```
/*
=====
Name          : DirectoryMonitor.cpp
Author       : Olympio Cipriano
Version      : 1.0
Copyright    :
Description  : CDirectoryMonitor implementation
=====
*/
```

Directory_Monitoring

```
#include "DirectoryMonitor.h"

CDirectoryMonitor::CDirectoryMonitor( DirectoryObserver* aObserver ) :
    (EPriorityStandard),
    iObserver( aObserver )
{
}

CDirectoryMonitor* CDirectoryMonitor::NewLC( DirectoryObserver* aObserver )
{
    CDirectoryMonitor* pNew = new ( ELeave ) CDirectoryMonitor( aObserver );
    CleanupStack::PushL( pNew );
    pNew->ConstructL();
    return pNew;
}

CDirectoryMonitor* CDirectoryMonitor::NewL( DirectoryObserver* aObserver )
{
    CDirectoryMonitor* pNew = CDirectoryMonitor::NewLC( aObserver );
    CleanupStack::Pop( pNew ); // self;
    return pNew;
}

void CDirectoryMonitor::ConstructL()
{
    ::LeaveIfError( iFs.Connect() );
    CActiveScheduler::Add( this ); // Add to scheduler
    mBuf = HBufC16::NewL(100);
}

CDirectoryMonitor::~CDirectoryMonitor()
{
    ()Cancel any request, if outstanding
    Close();
    iFileChangeCancel();
    if( iOldDir )
        delete iOldDir;
    if( iNewDir )
        delete iNewDir;
    if( iPath )
        delete iPath;
}

void CDirectoryMonitor::DoCancel()
{
    iFs.NotifyChangeCancel( iStatus );
}

void CDirectoryMonitor::Monitor( const TDesC& aDirPath )
{
    *iPath = aDirPath;
    ()Cancel any request, just to be sure
    =iStatus;
    NotifyChange( ENotifyFile, iStatus, *iPath );
    ListOldDir();
    SetActiveScheduler(); // scheduler a request is active
}

void CDirectoryMonitor::RunL()
{
    TInt::Set();
}
```

Directory_Monitoring

```
if ( err == KErrNone )
{
    ( iNewDir->ListFiles
      DiffDirectoryContents
if( iObserver )
for (int i = 0; i < iFileChangeArray.Count(); ++i)
{
    = iFileChangeArray[i].changeEntry
    ->DirectoryChanged( changeServerfileEntry.iName, changeEntry.action );
}
    ( );      Clean
    NotifyChangeIfNotifyEntry, iStatus, *iPath );
    ( ); // SetAsActive scheduler a request is active
}
}

TInt CDirectoryMonitor::RunError(TInt aError)
{
return aError;
}

void CDirectoryMonitor::ListFiles( CDir*& aDir )
{
if( aDir )
{
delete aDir;
    = 0;      aDir
}
    TEntry* GetDir( *iPath, KEntryAttNormal, ESortByName, aDir );
if( err )
{
//TODO: error handling
}

}

void CDirectoryMonitor::DiffDirectoryContents( )
{
for (int i = 0; (iOldDir != NULL) && (i < iNewDir->Count()); ++i)
{
const TEntry entry = (const TEntry& )(*iNewDir)[i];
if( ! SearchEntry( entry, iOldDir ) )
{
//file added
            (entryTEFileAdded) change
            Append(iFileChangeArray.
}
}
for (int i = 0; ( iOldDir != NULL ) && ( i < iOldDir->Count() ); ++i)
{
const TEntry entry = (const TEntry& )(*iOldDir)[i];
if( ! SearchEntry( entry, iNewDir ) )
{
//file added
            (entryTEFileChanged) change
            Append(iFileChangeArray.
}
}
}

bool CDirectoryMonitor::SearchEntry( const TEntry& aEntry, CDir * aDir )
{

```

Directory_Monitoring

```
for (int i = 0; ( aDir != NULL ) && ( i < aDir->Count() ); ++i)
{
    const TEntry& dirEntry = (const TEntry& ) (*aDir)[i];
    & entryNameTBe(CDesC&) dirEntry.iName;
    if ( entryName.Compare( aEntry.iName ) == 0 )
        return true;
}
return false;
}

void CDirectoryMonitor::Clean()
{
    if( iNewDir )
    {
        delete iNewDir;
        iNewDir = 0;
    }
    iFileChanBeArray;
    ListOldDir );
}
//end line
```

Postconditions

This code monitors a specific folder. A call to Monitor member function starts the monitoring process. The changes to be monitored are those occurred after the call to Monitor function.

Test application and other attachments

Directory Monitor Source Code: [File:DirectoryMonitoring.zip](#)