Archived:Playing multi-channel audio using CMdaAudioOutputStream

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 demonstrates how to play multi-channel audio using `CMdaAudioOutputStream`. This quite extensive code example is summarized below:

- **TSample**: `TSample` is a class for a sound sample. The class supports 16-bit samples with definable loops. If a loop is defined, the whole sample is first played once, and then the repeat area (`iRepStart` through `iRepEnd`) is repeated.
- **TAudioShared**: The `TAudioShared` class is used for data transfer and communication between threads. Among other things, the class takes care of synchronization issues between the threads.
- **MyActive**: The `MyActive` class is used for informing the requestor about user or timer activity.
- **CMixerThread**: The `CMixerThread` class takes care of mixing a maximum of 16 channels of sampled sound into audio output stream. Messages between the client thread and the mixer thread are handled via exceptions (`HandleExceptionL()` function).
- **CSndMixer**: `CSndMixer` is the main class for the user to produce sound. The class creates a second thread where the main mixer resides. Messages to the second thread are sent via exceptions (`SendCmd()` function).
- **CWavLoader**: The `CWavLoader` class loads 8-bit mono wav files into a TSample instance. The class cannot separate WAV header information from the proper audio data, so it is recommended that only files containing raw PCM data are loaded with this class. Otherwise, audible distortion may appear.

This snippet can be self-signed.

Usage example:

```
// Load a sound file
CWavLoader* wavLoader = CWavLoader::NewLC();
_LIT(KSampleFile, "C:\\Data\\Sounds\\Digital\\sample.wav");
iSample = wavLoader->LoadL(KSampleFile());
CleanupStack::PopAndDestroy(wavLoader);

// Create the sound mixer
iSndMixer = CSndMixer::NewL();
// Play the sample in channel 0 with frequency of 16,000 Hz and volume of 20
iSndMixer->Play(iSample, 0, 16000, 20);
```

MMP file

This snippet requires the following libraries:

```
LIBRARY mediaclientaudiostream.lib
```

Header file: TSample.h

```
#include <e32std.h>

class TSample
    {
```


(C) Copyright Nokia 2014. All rights reserved.

```
    public:
        inline TSample() :
            iData(NULL)
            {}

        inline TSample(TInt16* aData, TInt aLength) :
            iData(aData),
            iLength(aLength),
            iRepStart(0),
            iRepEnd(0)
            {}

        inline TSample(TInt16* aData, TInt aLength, TInt aRepStart,
                TInt aRepEnd) :
            iData(aData),
            iLength(aLength),
            iRepStart(aRepStart),
            iRepEnd(aRepEnd)
            {}

        TInt16* iData;
        TInt iLength;
        TInt iRepStart;
        TInt iRepEnd;
     };
```

Header file: TAudioShared.h

```
#include <e32std.h>

#include "TSample.h"

const TInt KMaxChannels = 16;
const TInt KAudioShift = 12;

enum TMixerCmd
    {
    ECmdStartMixer = 0,
    ECmdStopMixer,
    ECmdDestroyMixer
    };

class TAudioShared
    {
    public:
        /**
         * For thread death signaling.
         */
        RSemaphore iAliveMutex;

        /**
         * For sample attribute change signaling.
         */
        RMutex iMutex;
```


(C) Copyright Nokia 2014. All rights reserved.

```
        // Sample attributes.
        TSample iSample[KMaxChannels];
        TInt iVolume[KMaxChannels];
        TInt iFrequency[KMaxChannels];
        TBool iPlayStarted[KMaxChannels];
        TExcType iExc;
        TRequestStatus* iStatusPtr;

        /**
         * Main volume.
         */
        TInt iMainVolume;

        /**
         * Command parameter.
         */
        TMixerCmd iCmd;
    };
```

Header file: MyActive.h

```
#include <e32std.h>

class CMixerThread;

class MyActive : public CActive
    {
    public:  // Constructors
        void Request();
        MyActive(CMixerThread* aThread);

    private:  // Methods from base classes
        void DoCancel();
        void RunL();

    private:  // Data
        CMixerThread* iThread;
    };
```

Source file: MyActive.cpp

```
#include <e32base.h>

#include "MyActive.h"
#include "CMixerThread.h"

// Constructs the AO and sets its priority
MyActive::MyActive(CMixerThread* aThread) :
    CActive(CActive::EPriorityStandard),
    iThread(aThread)
    {
    // Adds the active object to the queue
    CActiveScheduler::Add(this);
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
// Cancels this task
void MyActive::DoCancel()
    {
    TRequestStatus* status = &iStatus;
    RThread().RequestComplete(status, KErrCancel);
    }

// Request accepting function
void MyActive::Request()
    {
    SetActive();
    iStatus = KRequestPending;
    }

// Informs the requestor that there is user activity or the timer is fired
void MyActive::RunL()
    {
    iThread->HandleExceptionL();
    Request();
    }
```

Header file: CMixerThread.h

```
#ifndef __CMIXERTHREAD_H_
#define __CMIXERTHREAD_H_

#include <E32Base.h>
#include <mdaaudiooutputstream.h>
#include <mda\common\audio.h>

#include "TAudioShared.h"

class MyActive;

class CMixerThread : CBase, MMdaAudioOutputStreamCallback
    {
    public:
        /**
         * Thread entry point.
         */
        static TInt ThreadFunction(TAny* aData);

        /**
         * Default destructor.
         */
        ~CMixerThread();

        /**
         * Exception handler.
         */
        void HandleExceptionL();

    private:  // Constructors
        /**
```

 (C) Copyright Nokia 2014. All rights reserved.

```
         * Default constructor.
         * @param aData data from the creator thread
         */
        CMixerThread(TAny* aData);

        /**
         * Two-phased constructor.
         * @param aData data from the creator thread
         */
        static CMixerThread* CreateL(TAny* aData);

        /**
         * Second phase constructor.
         */
        TInt Construct();

        /**
         * Second phase constructor.
         */
        void ConstructL();

    private:  // New functions
        /**
         * Starts the mixer.
         */
        void StartMixerL();

        /**
         * Stops the mixer.
         */
        void StopMixer();

        /**
         * Fill mixing buffer with new data.
         */
        void FillBufferL();

    private:  // Methods from base classes

        /**
         * From MMdaAudioOutputStreamCallback.
         */
        void MaoscPlayComplete(TInt aError);

        /**
         * From MMdaAudioOutputStreamCallback.
         */
        void MaoscBufferCopied(TInt aError, const TDesC8& aBuffer);

        /**
         * From MMdaAudioOutputStreamCallback.
         */
        void MaoscOpenComplete(TInt aError);

    private:  // Data
        CTrapCleanup* iCleanupStack;
        CActiveScheduler* iActiveScheduler;
```


(C) Copyright Nokia 2014. All rights reserved.

```
        CMdaAudioOutputStream* iStream;
        TMdaAudioDataSettings iSettings;

        TInt16* iBuffer;  // Buffer to CMdaAudioOutput
        TInt* iMixBuffer; // 32-bit buffer to mixing
        TPtrC8 iBufferPtr; // Pointer to iBuffer

        TAudioShared& iShared;  // Reference to shared data with client
        MyActive* iActive;

        // Current sample data pointers
        TInt16* iAudioData[KMaxChannels];

        // These are shifted by KAudioShift
        TInt iAudioPos[KMaxChannels];
        TInt iAudioEnd[KMaxChannels];
        TInt iRepStart[KMaxChannels];
        TInt iRepEnd[KMaxChannels];
    };

#endif /*__CMIXERTHREAD_H_*/
```

Source file: CMixerThread.cpp

```
#include <e32svr.h>

#include "CMixerThread.h"
#include "MyActive.h"
#include "TSample.h"
#include "TAudioShared.h"

const TInt KSampleRate = 16000;             // sample rate used
const TInt KBufferSize = KSampleRate / 20;  // 20 buffers per second

TInt CMixerThread::ThreadFunction(TAny* aData)
    {
    TAudioShared& shared = *((TAudioShared*)aData);

    // Tell the client we're alive
    shared.iAliveMutex.Wait();
    CMixerThread* mixerThread = NULL;

    TRAPD(error, mixerThread = CMixerThread::CreateL(aData));
    // TODO: Error handling

    shared.iStatusPtr = &(mixerThread->iActive->iStatus);

    // If we're still here, the active scheduler has been constructed.
    // Start the wait loop which runs until it's time to end the thread.
    CActiveScheduler::Start();
    delete mixerThread;

    // Tell the owning thread that it's safe to exit
    shared.iAliveMutex.Signal();
```


(C) Copyright Nokia 2014. All rights reserved.

```
    return KErrNone;
    }

CMixerThread* CMixerThread::CreateL(TAny* aData)
    {
    CMixerThread* self = new (ELeave) CMixerThread(aData);
    // TODO: Error handling (self may be NULL)

    TInt constructErr = self->Construct();
    // TODO: Error handling (constructErr)

    return self;
    }

TInt CMixerThread::Construct()
    {
    iCleanupStack = CTrapCleanup::New();

    TInt err = KErrNone;
    TRAP(err, ConstructL());
    return err;
    }

void CMixerThread::ConstructL()
    {
    // Create the active scheduler
    iActiveScheduler = new (ELeave) CActiveScheduler;
    CActiveScheduler::Install(iActiveScheduler);

    // Sound inits
    iSettings.iChannels = TMdaAudioDataSettings::EChannelsMono;
    iSettings.iSampleRate = TMdaAudioDataSettings::ESampleRate16000Hz;
    iSettings.iVolume = 1;

    iMixBuffer = new (ELeave) TInt[KBufferSize];
    iBuffer = new (ELeave) TInt16[KBufferSize];

    iBufferPtr.Set(TPtrC8((TUint8*)iBuffer, KBufferSize * 2));

    iActive = new (ELeave) MyActive(this);
    iActive->Request();
    }

CMixerThread::CMixerThread(TAny* aData) :
    iShared(*((TAudioShared*)aData))
    {
    }

CMixerThread::~CMixerThread()
    {
    delete iStream;
    delete iBuffer;
    delete iMixBuffer;
    delete iActiveScheduler;
    delete iCleanupStack;
    if (iActive)
        {
```


(C) Copyright Nokia 2014. All rights reserved.

```
        iActive->Cancel();
        }
    delete iActive;
    }

void CMixerThread::HandleExceptionL()
    {
    switch (iShared.iExc)
        {
        case EExcUserInterrupt: // Command from client
            {
            switch (iShared.iCmd)
                {
                case ECmdStartMixer:
                    {
                    StartMixerL();
                    break;
                    }
                case ECmdStopMixer:
                    {
                    StopMixer();
                    break;
                    }
                case ECmdDestroyMixer:
                    {
                    CActiveScheduler::Stop(); // Exit
                    break;
                    }
                }
            break;
            }
        default:
            {
            // On unknown exception, just exit this thread
            CActiveScheduler::Stop(); // Exit
            break;
            }
        }
    }

void CMixerThread::StartMixerL()
    {
    iStream = CMdaAudioOutputStream::NewL(*this);
    iStream->Open(&iSettings);
    }

void CMixerThread::StopMixer()
    {
    iStream->Stop();
    delete iStream;
    iStream = NULL;
    }

void CMixerThread::FillBufferL()
    {
    // Wait for access to shared data
```


(C) Copyright Nokia 2014. All rights reserved.

```
    iShared.iMutex.Wait();

    TInt volume = iShared.iMainVolume;

    // Gather new sample information
    for (TInt i = 0; i < KMaxChannels; i++)
        {
        if (iShared.iPlayStarted[i])
            {
            iShared.iPlayStarted[i] = EFalse;
            TSample& sample = iShared.iSample[i];

            iAudioData[i] = sample.iData;
            iAudioPos[i] = 0;
            iAudioEnd[i] = sample.iLength << KAudioShift;
            iRepStart[i] = sample.iRepStart << KAudioShift;
            iRepEnd[i] = sample.iRepEnd << KAudioShift;
            }
        }
    // Give access to shared data
    iShared.iMutex.Signal();

    // Clear the buffer. This has to be done because channels are mixed by
    // adding their values to each other
    Mem::FillZ(iMixBuffer, KBufferSize * 4);

    // Mix active channels
    for (TInt i = 0; i < KMaxChannels; i++)
        {
        if (iAudioData[i] != NULL)
            {
            TInt* buf = iMixBuffer;
            TInt* bufEnd = buf + KBufferSize;

            TInt16* src = iAudioData[i];

            TInt pos = iAudioPos[i];
            TInt posEnd = iAudioEnd[i];
            TInt repStart = iRepStart[i];
            TInt repEnd = iRepEnd[i];
            TInt posAdd = (iShared.iFrequency[i] << KAudioShift) / KSampleRate;
            TInt volume = iShared.iVolume[i];

            while (buf < bufEnd)
                {
                TInt sample = (src[pos >> KAudioShift] * volume);
                pos += posAdd;
                if (pos >= posEnd)
                    {
                    if (repEnd == 0)
                        {
                        iAudioData[i] = NULL;
                        break;
                        }
                    else
                        {
                        pos = repStart;
```


(C) Copyright Nokia 2014. All rights reserved.

```
                        posEnd = repEnd;
                        }
                    }
                *buf++ += sample;
                }
            iAudioPos[i] = pos;
            iAudioEnd[i] = posEnd;
            }
        }

    // Convert 32-bit mixing buffer to 16-bit output buffer
    TInt* buf = iMixBuffer;
    TInt* bufEnd = buf + KBufferSize;
    TInt16* buf2 = iBuffer;
    while (buf < bufEnd)
        {
        // 32-bit mixer buffer contents are multiplied by main volume.
        // Shifts are in two phases to prevent overflow and to maintain
        // quality.
        TInt value = ((*buf++ >> 8) * volume) >> 8;

        // Prevent sound from trashing on overboost volume
        if (value < -0x7FFF)
            value = -0x7FFF;
        if (value > 0x7FFF)
            value = 0x7FFF;

        // and write to buffer
        *buf2++ = (TInt16)value;
        }

    // Write 16-bit buffer to CMdaAudioOutputStream
    iStream->WriteL(iBufferPtr);
    }

void CMixerThread::MaoscBufferCopied(TInt aError, const TDesC8& aBuffer)
    {
    // TODO: Error handling (aError)

    TRAPD(error, FillBufferL());
    // TODO: Error handling (error)
    }

void CMixerThread::MaoscOpenComplete(TInt aError)
    {
    // TODO: Error handling (aError)

    iStream->SetVolume(iStream->MaxVolume());
    TRAPD(error, FillBufferL());
    // TODO: Error handling (error)
    }

void CMixerThread::MaoscPlayComplete(TInt aError)
    {
    // TODO: Error handling (aError)
```


(C) Copyright Nokia 2014. All rights reserved.

```
    iStream->SetVolume(iStream->MaxVolume());
    TRAPD(error, FillBufferL());
    // TODO: Error handling (error)
    }
```

Header file: CSndMixer.h

```
#ifndef __CSNDMIXER_H_
#define __CSNDMIXER_H_

#include <E32Base.h>

#include "TAudioShared.h"
#include "TSample.h"

class CSndMixer : public CBase
    {
    // Constructors and destructor omitted for brevity
    // ...

    public:  // New functions
        /**
         * Plays a sample with given parameters.
         * @param aSample a sample to play
         * @param aChannel channel 0-15 to play the sample on
         * @param aVolume volume to play the sample with
         */
        void Play(const TSample& aSample, TInt aChannel, TInt aFrequency,
                TInt aVolume = 256);

    private:  // New functions
        /**
         * Sends a command to the mixer thread (CMixerThread)
         * @param aCmd a command to send
         */
        void SendCmd(TMixerCmd aCmd);

    private:  // Data
        TAudioShared iShared;   // Shared data with mixer thread
        RThread iMixerThread;   // Handle to mixer thread
    };

#endif /*__CSNDMIXER_H_*/
```

Source file: CSndMixer.cpp

```
#include <e32svr.h>

#include "CSndMixer.h"
#include "CMixerThread.h"

_LIT(KMixer, "Mixer");

void CSndMixer::ConstructL()
```


(C) Copyright Nokia 2014. All rights reserved.

```
    {
    iShared.iMainVolume = 50;   // Default main volume

    User::LeaveIfError(iShared.iAliveMutex.CreateLocal(1));
    User::LeaveIfError(iShared.iMutex.CreateLocal());
    User::LeaveIfError(iMixerThread.Create(KMixer,
            CMixerThread::ThreadFunction,
            KDefaultStackSize,
            KMinHeapSize,
            KMinHeapSize + 1000000,
            &iShared));

    // Give all possible priority to audio
    iMixerThread.SetPriority(EPriorityRealTime);
    iMixerThread.Resume();
    }

// Other constructors omitted for brevity
// ...

CSndMixer::~CSndMixer()
    {
    SendCmd(ECmdDestroyMixer);
    iShared.iAliveMutex.Wait();
    iShared.iAliveMutex.Close();
    iShared.iMutex.Close();
    }

void CSndMixer::Play(const TSample& aSample, TInt aChannel, TInt aFrequency,
        TInt aVolume)
    {
    iShared.iMutex.Wait();
    iShared.iPlayStarted[aChannel] = ETrue;
    iShared.iSample[aChannel] = aSample;
    iShared.iFrequency[aChannel] = aFrequency;
    iShared.iVolume[aChannel] = aVolume;
    iShared.iMutex.Signal();

    SendCmd(ECmdStartMixer);
    }

void CSndMixer::SendCmd(TMixerCmd aCmd)
    {
    iShared.iMutex.Wait();
    iShared.iCmd = aCmd;
    iShared.iMutex.Signal();
    iShared.iExc = EExcUserInterrupt;

    TRequestStatus* status = iShared.iStatusPtr;
    if (status->Int() == KRequestPending)
        {
        iMixerThread.RequestComplete(status, KErrNone);
        }
    }
```

 (C) Copyright Nokia 2014. All rights reserved.

Header file: CWavLoader.h

```
#ifndef __CWAVLOADER_H_
#define __CWAVLOADER_H_

#include <e32base.h>

#include "TSample.h"

class CWavLoader : public CBase
    {
    // Constructors and destructor omitted for brevity
    // ...

    public:
        /**
         * Loads a wav file and creates a sample from it.
         * @param aFileName path and file name of the wav file
         */
        TSample LoadL(const TFileName& aFileName);
    };

#endif /*__CWAVLOADER_H_*/
```

Source file: CWavLoader.cpp

```
#include <aknutils.h>
#include <e32svr.h>
#include <f32file.h>

#include "CWavLoader.h"

// Constructors and destructor omitted for brevity
// ...

/**
 * Creates a TSample from a file.
 */
TSample CWavLoader::LoadL(const TFileName& aFileName)
    {
    TFileName name(aFileName);

    // Connect to the file server session and open the file for reading
    RFs fsSession;
    User::LeaveIfError(fsSession.Connect());
    RFile file;
    CleanupClosePushL(fsSession);
    CleanupClosePushL(file);
    User::LeaveIfError(file.Open(fsSession, name, EFileStream | EFileRead));

    // Store the file size
    TInt size;
    User::LeaveIfError(file.Size(size));
```


(C) Copyright Nokia 2014. All rights reserved.

```
    // Store the file contents
    TUint8* tbuf = new (ELeave) TUint8[size];
    CleanupStack::PushL(tbuf);

    TPtr8 ptr((TUint8*)tbuf, size);
    User::LeaveIfError(file.Read(ptr));

    TInt16* buf = new (ELeave) TInt16[size];
    for (TInt i = 0; i < size; i++)
        {
        TInt v = tbuf[i];
        v -= 128;
        buf[i] = (TInt16)(v * 256);
        }

    CleanupStack::PopAndDestroy(3); // tbuf, file, fsSession

    // Create the TSample
    TSample sample;
    sample.iData = buf;
    sample.iLength = size;
    sample.iRepStart = 0;
    sample.iRepEnd = 0;

    return sample;
    }
```

Postconditions

A wav file is played with the parameters given to `CSndMixer::Play()`.

See also

- Multi-channel audio playback
- S60 Platform: Sound Mixer Example

(C) Copyright Nokia 2014. All rights reserved.