Playing PCM audio streams

The `CStreamPlayer` illustrates how to play PCM encoded audio streams using `CMdaAudioOutputStream`. The audio needs to be PCM encoded data supplied via byte buffer. If you want to use other formats you need to convert the data to PCM before playing it with the `CMdaAudioOutputStream`.

`CMdaAudioOutputStream` requires the player class to implement `MMdaAudioOutputStreamCallback` callback interface. The `MaoscOpenComplete` method is called when the construction for the output stream is completed. Playing should never be started until this method has been called by the output stream object.

As shown in the example implementation for the `MaoscOpenComplete` after setting the final settings the stream is played by calling the *WriteL* method of the output stream. Calling the method with the PCM buffer will cause the output stream to play the buffer, and after the buffer is played `MaoscBufferCopied` will be called. Then a new buffer is set for playing by calling the `WriteL` method again and also new data is copied to be ready for the next call of the `MaoscBufferCopied`' method.

Remarks:

1. Note this Known Issue about the function of *MaoscPlayComplete*. Essentially it is not getting called if the stream runs out of buffers on the device (though, confusingly, it is called on the emulator). So you need other workarounds to find out when the last sample of your stream has played.
2. Keeping the stream open after the last buffer has been played, rather than calling Stop() or destroying it, will leave the audio hardware turned on. This will lead to a severe reduction in battery life on most devices. So you should make sure to stop the stream if you are not planning to play audio for more than a few seconds.

Stream_Play.cpp

```
CStreamPlayer::CStreamPlayer(void)
:iVolume(10),iReadSize(0),iPlayError(KErrNone)
    {
    }

CStreamPlayer::~CStreamPlayer()
{
 Stop();

 delete iStream;
 iStream = NULL;

 delete iPlayBuffer;
 iPlayBuffer = NULL;

    iBuffer.ResetAndDestroy();
}

void CStreamPlayer::PlayL(const TDesC8& aPlayBuffer)
{
 Stop();
 iReadSize = 0;

 delete iPlayBuffer;
 iPlayBuffer = NULL;

 delete iStream;
 iStream = NULL;

 iBuffer.ResetAndDestroy();
```


(C) Copyright Nokia 2014. All rights reserved.

```
 iPlayBuffer = HBufC8::NewL(aPlayBuffer.Length());
 iPlayBuffer->Des().Copy(aPlayBuffer);

 TDes8* buffer = new(ELeave) TBuf8<KPcmBufferSize>;
 buffer->SetMax();
 CleanupStack::PushL(buffer);
 User::LeaveIfError(iBuffer.Append(buffer));
 CleanupStack::Pop(buffer);

 buffer = new(ELeave) TBuf8<KPcmBufferSize>;
 buffer->SetMax();
 CleanupStack::PushL(buffer);
 User::LeaveIfError(iBuffer.Append(buffer));
 CleanupStack::Pop(buffer);

 iReadSize  = 0;
 iCurrIndex = 0;
    for (TInt index = 0; index < iBuffer.Count(); index++)
 {
  ReadFromFromL(index);
 }

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

void CStreamPlayer::Stop()
{
 if(iStream)
  iStream->Stop();
}


void CStreamPlayer::SetVolume(TInt aVol)
{
 iVolume = aVol;

 if(iVolume < 0)
  iVolume = 0;
 else if(iVolume > 10)
  iVolume = 10;

 if(iStream)
 {
 iStream->SetVolume(((iStream->MaxVolume() * iVolume) / 10));
 }
}

void CStreamPlayer::MaoscOpenComplete(TInt aError)
{
 iPlayError = aError;

 if (aError==KErrNone && iStream)
 {
 iSettings.iSampleRate = TMdaAudioDataSettings::ESampleRate8000Hz;
 iSettings.iChannels = TMdaAudioDataSettings::EChannelsMono;
 //iSettings.iVolume = ((iStream->MaxVolume() * iVolume) / 10);
```


(C) Copyright Nokia 2014. All rights reserved.

```
  iStream->SetAudioPropertiesL(iSettings.iSampleRate,
iSettings.iChannels);
 SetVolume(iVolume);
 iStream->SetPriority(EPriorityNormal, EMdaPriorityPreferenceNone);

        iStream->WriteL(*iBuffer[0]);
 }
}

void CStreamPlayer::MaoscBufferCopied(TInt aError, const TDesC8& aBuffer)
{
 iPlayError = aError;

 if(aError == KErrNone && iStream)
 {
  if (&aBuffer == iBuffer[0])
  {
   iCurrIndex = 0;

   if((*iBuffer[1]).Length())
   {
    iStream->WriteL(*iBuffer[1]);
    ReadFromFromL(0);
   }
  }
  else// if (&aBuffer == iBuffer[1])
  {
   iCurrIndex = 1;

   if((*iBuffer[0]).Length())
   {
    iStream->WriteL(*iBuffer[0]);
    ReadFromFromL(1);
   }
  }
 }
}

void CStreamPlayer::MaoscPlayComplete(TInt aError)
{
 iPlayError = aError;

  if (aError == KErrUnderflow)
  {
   if(iPlayBuffer->Des().Length() > iReadSize)
   {
    iCurrIndex = 0;
     for (TInt index = 0; index < iBuffer.Count(); index++)
   {
    ReadFromFromL(index);
   }

   iStream->WriteL(*iBuffer[0]);
  }
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
}


void CStreamPlayer::ReadFromFromL(TInt aBufferIndex)
{
 if((aBufferIndex == 0 || aBufferIndex == 1) && iPlayBuffer)
 {
  (*iBuffer[aBufferIndex]).FillZ();
  (*iBuffer[aBufferIndex]).Zero();

  if((iPlayBuffer->Des().Length() > iReadSize))
  {
  TInt ReadFor(KPcmBufferSize);
  if(iPlayBuffer->Des().Length() < (iReadSize + ReadFor))
   {
   ReadFor = iPlayBuffer->Des().Length()  - iReadSize;
   }
  iBuffer[aBufferIndex].Copy(iPlayBuffer->Des().Mid(iReadSize,ReadFor));
  iReadSize = iReadSize + ReadFor;
  }
 }
}
```

Stream_Play.h

```
#include <mda\common\audio.h>
#include <Mda\Client\Utility.h>
#include <Mda\Common\Resource.h>
#include <MdaAudioOutputStream.h>
#include <mmf\server\mmfdatabuffer.h>

const TInt KPcmBufferSize = 10000;

class CStreamPlayer : public CBase, public MMdaAudioOutputStreamCallback
 {
public:
 CStreamPlayer(void);
 ~CStreamPlayer();
public:
 void PlayL(const TDesC8& aPlayBuffer);
    void Stop();
    TInt Volume(void){ return iVolume; }
    void SetVolume(TInt aVol);
protected:
   virtual void MaoscOpenComplete(TInt aError);
   virtual void MaoscBufferCopied(TInt aError, const TDesC8& aBuffer);
   virtual void MaoscPlayComplete(TInt aError);
private:
    void ReadFromFromL(TInt aBufferIndex);
private:
    TInt
 iVolume,iReadSize,iCurrIndex,iPlayError;
 RPointerArray<TDes8>  iBuffer;
 CMdaAudioOutputStream*  iStream;
 TMdaAudioDataSettings  iSettings;
```


(C) Copyright Nokia 2014. All rights reserved.

```
HBufC8*   iPlayBuffer;
TBool   iPlaying;
};
```


(C) Copyright Nokia 2014. All rights reserved.