g729 codec

By **carl morey**

Introduction to directsound and the g729 codec using visual basic

Version: **3 (See All)**
Posted: **19 May 2009**
Views: **353**
Bookmarked: **2 times**
Unedited contribution

Get Article's HTML | Modify | Delete

ANNOUNCEMENTS
Smart Client Comp
Monthly Competition

CHAPTERS
Desktop Development
Web Development
Enterprise Systems
Multimedia
Database
Platforms, Frameworks & Libraries
Languages
General Programming
Algorithms & Recipes
Bugs & Workarounds
Collections
Cryptography & Security
Date and Time
DLLs & Assemblies
Exception Handling
Localisation
Macros and Add-ins
Programming Tips
String handling
Internet / Network
Threads, Processes & IPC
WinHelp / HTMLHelp
Graphics / Design
Development Lifecycle
General Reading
Third Party Products

SERVICES
Product Catalog
Code-signing Certificates
Job Board
CodeProject VS2008 Addin

FEATURE ZONES
Product Showcase
WhitePapers / Webcasts
ASP.NET Web Hosting

Search Articles Go! Advanced Search Add to IE Search

Print Report Watch Bookmark Share Discuss Email

1 vote for this article.
Popularity: 0.00 Rating: **5.00** out of 5
1 2 3 4 5

Download programs.zip - 260.04 KB

Introduction

I have used microsoft live messenger to communicate using voice through the sound card on my computer. Using net monitor I found the following:

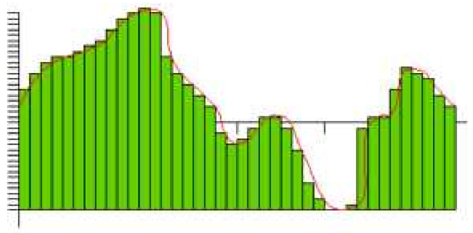
Rtp: PayloadType = G723 Audio, 8000Hz [1 Channel], SSRC = 3551173066, Seq = 10633

Using task manager I could see that the process was consuming 50% of CPU time. So it uses:

- RTP internet protocol
- g723 voice codec
- 8000 Hz sample rate
- Some heavy cpu calculations

A good way to learn about a process is to write a program to to it. I have started and this article describes g729 and direct sound using visual basic. I have included links to sites which I have used in this process.

Pulse code modulation



An analog-to-digital converter measures sound waves at frequent intervals.

ADCs and DACs
Imagine using your computer to record yourself talking. First, you speak into a microphone that you have plugged into your sound card. The ADC translates the analog waves of your voice into digital data that the computer can understand. To do this, it samples, or digitizes, the sound by taking precise measurements of the wave at frequent intervals.
If you were to play your recording back through the speakers, the DAC would perform the same basic steps in reverse. With accurate measurements and a fast sampling rate, the restored analog signal can be nearly identical to the original sound wave.

When you sample the wave with an analog-to-digital converter, you have control over two variables:

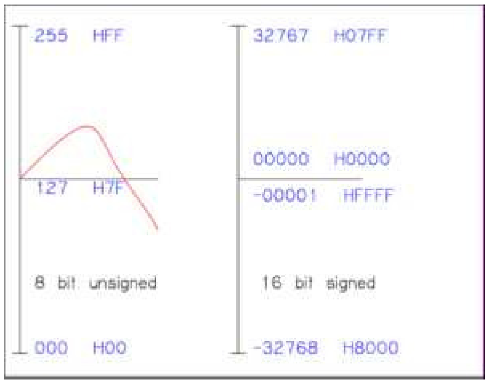
The sampling rate - Controls how many samples are taken per second
The sampling precision - Controls how many different gradations (quantization levels) are possible when taking the sample

In the case of CD sound, fidelity is an important goal, so the sampling rate is 44,100 samples per second and the number of gradations is 65,536. At this level, the output of the DAC so closely matches the original waveform that the sound is essentially "perfect" to most human ears.

44,100 samples/(channel*second) * 2 bytes/sample * 2 channels * 74 minutes * 60 seconds/minute = 783,216,000 bytes

This was the basis of the origional cd's. The raw pcm with a header with number of bytes
ect makes up a .wav file. The audio codec MP3 allows compression of CD-quality audio files by a factor of 12 with little loss in quality.

With voice you can have narrow band 8000 samples/sec or wide band 16000 samples/sec.
You can have 256 levels ie one unsigned byte. I have tried this it's supported by direct sound
but only one codec speex can use it. I did not use it in my program as the limited dynamic range means that the background noise from the microphone is very annoying.
65536 ie 2 bytes(signed 16 bit integer) is used by all codecs. This produces a much better result. The data is stored little endian in a byte array ie least significant byte first.

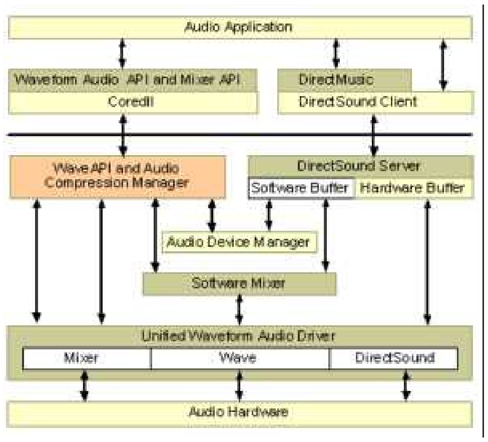


Direct sound

Microsoft introduced its DirectX standard in Windows 95. DirectSound offered basic stereo left and right panning effects.

DirectSound itself acts as a sound-mixing engine, using system RAM to hold the different audio streams in play.

DirectX is a set of low-level APIs for creating games and other high-performance multimedia applications. It includes support for high-performance 2D and 3D graphics, sound, and input.
DirectSound implements a model for playing and capturing digital sound samples and mixing sample sources. Like other elements of the DirectX API, DirectSound uses the hardware to its greatest advantage whenever possible, and it emulates hardware features in software when the feature is not present in the hardware.DirectSound implements a model for playing and capturing digital sound samples and mixing sample sources. Like other elements of the DirectX API, DirectSound uses the hardware to its greatest advantage whenever possible, and it emulates hardware features in software when the feature is not present in the hardware.
The following illustration shows the relationships between DirectSound and other system audio components.



DirectSound and the standard Windows waveform-audio functions provide alternative paths to the waveform-audio portion of the sound hardware. A single device provides access from one path at a time. If a waveform-audio driver has allocated a device, an attempt to allocate that same device by using DirectSound will fail. Similarly, if a DirectSound driver has allocated a device, an attempt to allocate the device by using the waveform-audio driver will fail.

The latest version is DirectX 2009. Download Microsoft DirectX SDK (DXSDK_Mar09.exe) from:

http://www.microsoft.com/downloads/details.aspx?FamilyID=24a541d6-0486-4453-8641-1eee9e21b282&displaylang=en

The SDK has example programs to play and capture sound. I worked directly from them to produce my code. Unfortunally they are all in C++ I dont really want to learn C++ well enough to write my program so I used reflection. Or rather Red gates .net reflector available from:

http://www.red-gate.com/products/reflector/

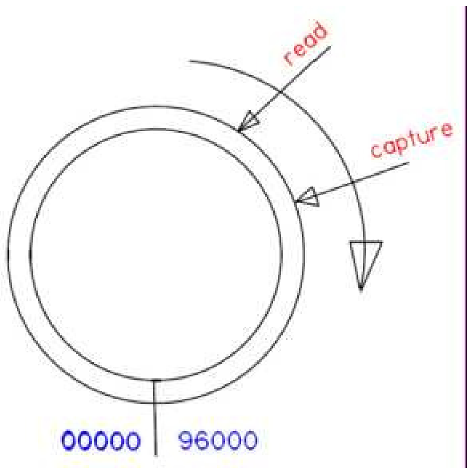
This can decompile a C++ .exe into visual basic. This works on the directx samples and any compiled C++ which has not been deliberatly obfuscated by the writer.

The code for a 6 second voice capture class:

☐ Collapse Copy Code

```
Imports System.Threading
Imports Microsoft.DirectX
Imports Microsoft.DirectX.DirectSound 'need to add these in project references
Public Class encoder
    Private recdevice As Capture  'again default device
    Private rbuffd As CaptureBufferDescription 'record buffer description
    Private inputformat As WaveFormat 'format of input wave
    Private cbuff As CaptureBuffer  'record audio buffer
    Private capdevguid As Guid = Guid.Empty 'use system guid for default capture
    Private capture As Thread = Nothing 'voice capture
    Public NotificationEvent As AutoResetEvent = Nothing
    Public applicationNotify As Notify = Nothing
    Private PositionNotify As BufferPositionNotify() = New BufferPositionNotify(1000) {}
    Private notifysize As Integer = 3200 '=0.2 seconds X 30 = 6 seconds
    Public Sub New()
        Try  'setup capture buffer
            recdevice = New Capture(capdevguid)
            rbuffd = New CaptureBufferDescription  'record buffer description
            rbuffd.BufferBytes = 96000  'size of buffer = 6 seconds
            inputformat.FormatTag = WaveFormatTag.Pcm 'must be pcm
            inputformat.SamplesPerSecond = 8000 'for voice only
            inputformat.BitsPerSample = 16 'only for voice
            inputformat.Channels = 1 'mono only
            inputformat.BlockAlign = 2 '2 bytes per sample
            inputformat.AverageBytesPerSecond = 16000 'must specify all
            rbuffd.Format = inputformat  'set buffer description
            cbuff = New CaptureBuffer(rbuffd, recdevice) 'we now have device and buffer
            NotificationEvent = New AutoResetEvent(False) 'not signalled yet
            For i = 0 To 29 'get autoreset every .2 seconds
                PositionNotify(i).Offset = (notifysize * (i + 1)) - 1
                PositionNotify(i).EventNotifyHandle = NotificationEvent.SafeWaitHandle.DangerousGetHandle
            Next
            applicationNotify = New Notify(cbuff)
            applicationNotify.SetNotificationPositions(PositionNotify, 30)
            cbuff.Start(True) 'the voice recorder now runs
        Catch err As Exception
            MsgBox("could not record audio")
        End Try
        capture = New Thread(AddressOf wait_thread) 'thread waits till got set value
        capture.IsBackground = True
        capture.Start()
        mainform.Button2.Enabled = False 'disable this button
        capture.Join()                   'until capture finished
        mainform.Button2.Enabled = True
        cbuff.Dispose() 'must dispose of capture buffer after finished
    End Sub
    Private Sub wait_thread()
        Dim nextoffset As Integer = 0 'we use our own pointer to capture buffer
        Try  'make a new file if one already exists
            System.IO.File.Delete("voice.pcm")
        Catch ex As Exception
        End Try
        Do
            NotificationEvent.WaitOne(-1, True) 'waits here till have 3200 bytes
            'Dim readpos As Integer
            'Dim capturepos As Integer
            'cbuff.GetCurrentPosition(capturepos, readpos)
            'Debug.WriteLine(capturepos & "  " & readpos)
            Dim data = cbuff.Read(nextoffset, GetType(Byte), LockFlag.None, New Integer() {notifysize})
            My.Computer.FileSystem.WriteAllBytes("voice.pcm", data, True)
            nextoffset = nextoffset + notifysize 'move our pointer on
            If nextoffset = rbuffd.BufferBytes Then nextoffset = 0
            If nextoffset = 0 Then Exit Sub 'can set to get any size recorded buffer
        Loop
    End Sub
    Public Shared Sub sound()
        Dim amplitude As Integer = 100 ' 0 to 100 zero to max amplitude
        Dim frequency As Double = 0.32 ' about 400 hz ratio for other frequencies
        Dim Datas = New [Byte](95999) {}
        Dim angle As Double = 0
        Dim t As Integer = 0
        For i = 0 To 47999 '2 bytes per sample
            Datas(2 * i) = 0
            t = amplitude * Math.Sin((angle + i) * frequency)
            If t < 0 Then t = t + 255 'this for negative values
            Datas(2 * i + 1) = t 'only the msbyte. The lsbyte is zero tone
        Next
        My.Computer.FileSystem.WriteAllBytes("voice.pcm", Datas, False)
    End Sub
End Class
```

A good way to understand the capture buffer (and the play with looping) is as a circular buffer:



I broke the buffer into 30 3200 byte segments because I am using this code for other purposes. You can also monitor the buffer if you uncomment the cbuff.getcurrentposition lines.

The play raw pcm class is almost identical to the capture code so I wont list it here. It uses a secondary buffer. The play buffer is not broken up and is just one 96000(6 sec) buffer.

It's worthwhile to mention a few quirks of directX.directsound.

The play buffer takes time to set up after setting the secondary buffer:

☐ Collapse Copy Code

```
sbuff = New SecondaryBuffer(sbuffd, appdevice)
```

If you immediately sbuff.Write or sbuff.play you may get an error that sbuff does not exist.
I have given it one second to be on the safe side(much less will proberly do).

If you reload the form to quickly after closing the form while running the program in debug
from the VB express program you may get an error:

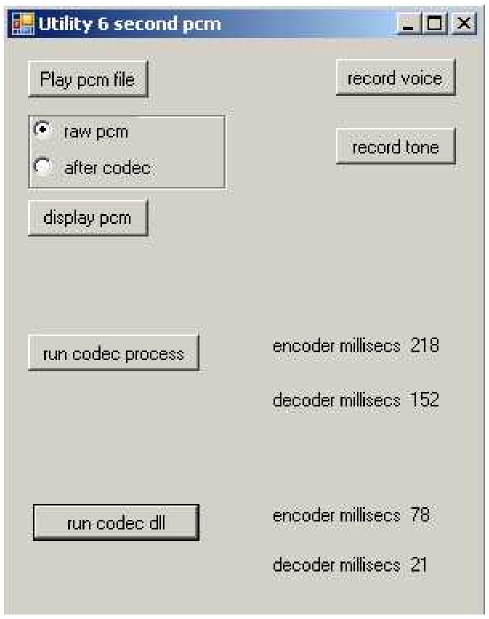
C:\WINDOWS\assembly\GAC\Microsoft.DirectX\1.0.2902.0__31bf3856ad364e35\Microsoft.DirectX.dll is attempting managed execution inside OS Loader lock. Do not attempt to run managed code inside a DllMain or image initialization function since doing so can cause the application to hang.

Just continue or try again it does not appear to make any difference to the program.

An explaination of loader lock is at:
http://blogs.msdn.com/cbrumme/archive/2003/08/20/51504.aspx

I believe it is because the directx .dll is not finalised before trying to restart. Ultimately, every OS process shuts down via a call to ExitProcess or TerminateProcess. ExitProcess is the nice orderly shutdown, which notifies each DLL of the termination. TerminateProcess is ruder, in that the DLLs are not informed. This is a very rare occurance and only happens inside the VB express program. I put the codec_utility .exe in an autostart program as an application and it never happens.

To allow simple user interface to the direct sound capture and play classes I have used a windows form. I also included user interfaces to the g729 codec codes:



encoder millisecs 218

decoder millisecs 152

encoder millisecs 78

decoder millisecs 21

Voice codecs

Directsound produces raw pcm. This can be compressed into a much smaller size.There are many audio compression codecs:

Audio compression ISO/IEC MPEG-1 Layer III (MP3) · MPEG-1 Layer II · MPEG-1 Layer I · AAC · HE-AAC ITU-T G.711 · G.718 · G.719 · G.722 · G.722.1 · G.722.2 · G.723 · G.723.1 · G.726 · G.728 · G.729 · G.729.1 · G.729a AC3 · AMR · Apple Lossless · ATRAC · FLAC · iLBC · Monkey's Audio · μ-law · Musepack · Nellymoser · OptimFROG · RealAudio · RTAudio · SHN · Siren · Speex · Vorbis · WavPack · WMA · TAK · True Audio

The first voice codec I looked at was speex available at:

http://www.speex.org/

This is speech codec that is open-source and free from software patent royalties.The Speex codec is designed to be very flexible and support a wide range of speech quality and bit-rate. Support for very good quality speech also means that Speex can encode wideband speech (16 kHz sampling rate) in addition to narrowband speech (telephone quality, 8 kHz sampling rate).

You can test it in my program that runs the process by subsituting the speex***.exe's for the g729. It can give a compression factor of 16 if you select quality 2.

It has a complete open source listing of all the api's. To use it in voip requires a good knowledge of C to use the api's.

Speex is a lossy codec, which means that it achives compression at the expense of fidelity of the input speech signal. Many of the more popular codecs in the software world are lossy, meaning that they reduce quality by some amount in order to achieve compression, but use some algorithm to create the impression of the data being there. speex and g729 use Conjugate Structure Algebraic Code Excited Linear Prediction.

I next looked at g729 which seems to be the latest comercial voice codec.The ITU's full g series codecs are available at:

http://www.itu.int/rec/T-REC-G/e

All of these codecs can be downloaded and used for your private purposes. I found the best source of the g729 codec to be voiceage:

http://www.voiceage.com/

The VoiceAge Open G.729 Implementation:

This free offer is a single-port Win32 implementation that allows for experimentation with the G.729 standard. You can take it from one desktop computer to another without constraints. It is important to note that this version is not indemnified, which means that this code cannot be used for commercial purposes — it is restricted to research and prototype development. If commercial deployment is planned, you must obtain the legal right by licensing the intellectual property.

The ITU-T G.729 standard provides to the telecommunications industry a low bit rate 8-kbps speech coding algorithm with toll quality, fulfilling today's quest for bandwidth, quality of service, and cost savings. Beyond its 8-kbps codec, the G.729 technology, with its annexes, covers a wide range of additional specifications such as voice activity detection and several other bit rates.

The download package gives:

va_g729a.lib Win32 statically linkable library of G729 floating-point object code for Pentium and compatible processors.
va_g729a.h API prototypes and constants declarations required by the sample programs.
va_g729a_encoder.c Encoder sample application demonstrating encoder API calls to the codec for encoding a speech file.
va_g729a_decoder.c Decoder sample application demonstrating decoder API calls to the codec for decoding a speech file.
va_g729a_encoder.exe Encoder sample program executable for the Win32 platform.
va_g729a_decoder.exe Decoder sample program executable for the Win32 platform.

The .exe applications can be used with process.start to demonstrate the codec operation.
They did not suit my application so I rewrote the C code to produce g729e.exe and g729d.exe applications:

The C++ for the encoder is given below:

☐ Collapse Copy Code

```
//file.h
#define  L_FRAME_COMPRESSED 10
#define  L_FRAME            80
void va_g729a_init_encoder();
void va_g729a_encoder(short *speech, unsigned char *bitstream);
void va_g729a_init_decoder();
void va_g729a_decoder(unsigned char *bitstream, short *synth_short, int bfi);
//file.cpp
#include "stdio.h"
#include "va_g729.h"
#include "va_g729.h"
void main(void)
{
 int nb_frame;
 FILE* fp_in;
 FILE* fp_out;
 short   speech[L_FRAME];
 unsigned char serial[L_FRAME_COMPRESSED];
 va_g729a_init_encoder();
    fp_in = fopen("voice.pcm", "rb");
    fp_out = fopen("code.pcm", "wb");
 nb_frame = 0;
 while (fread(speech, sizeof(short), L_FRAME, fp_in) == L_FRAME)
 {
  va_g729a_encoder(speech, serial);
  fwrite(serial, sizeof(char), L_FRAME_COMPRESSED, fp_out);
 }
 fclose(fp_out);
 fclose(fp_in);
}
```

Using process.start on my xp computer requires a minimun of 100 ms. To use this in voip where I use 20 msec packets is quite impossible. We have to use the C api's directly. This is fine if you are using C however I want to use visual basic. The api's are in va_g729a.lib which is inaccessable from visual basic.

Convert .lib to .dll

A dynamic-link library (DLL) is an executable file that acts as a shared library of functions. Dynamic linking provides a way for a process to call a function that is not part of its executable code. The executable code for the function is located in a DLL, which contains one or more functions that are compiled, linked, and stored separately from the processes that use
them.

You can also create your own libraries of classes and algorithms that any application can use. With Visual C++, you can create three kinds of libraries:

- Dynamic link libraries (.dll with entry points).
- Static libraries.(.lib)
- Managed assemblies.(.dll as a .net assembly)

I wrote a C++ progam to compile a managed asembly(.dll) which can be accessed from visual basic. The code uses the api's from the static library va_g729a.lib.

The C++ for the codec.dll is given below:

☐ Collapse Copy Code

```
// files.h
#pragma once
#pragma managed(push, off)
extern "C" {
#include "va_g729.h"
}
#pragma managed(pop)
using namespace System;
namespace masm
{
 public ref class math
 {
     public:
     void encoder(void);
     void decoder(void);
 };
}
```

☐ Collapse Copy Code

```
//files.cpp
#include <stdio.h>
#include "stdafx.h"
#include "codec.h"
#pragma managed(push, off)
extern "C" {
#include "va_g729.h"
}
#pragma managed(pop)
namespace masm
{
 void math::encoder(void)
 {
    int nb_frame;
 FILE* fp_in;
 FILE* fp_out;
 short   speech[L_FRAME];
 unsigned char serial[L_FRAME_COMPRESSED];
 va_g729a_init_encoder();
    fp_in = fopen("voice.pcm", "rb");
    fp_out = fopen("code.pcm", "wb");
 nb_frame = 0;
 while (fread(speech, sizeof(short), L_FRAME, fp_in) == L_FRAME)
 {
  va_g729a_encoder(speech, serial);
  fwrite(serial, sizeof(char), L_FRAME_COMPRESSED, fp_out);
 }
 fclose(fp_out);
 fclose(fp_in);
 }
 void math::decoder(void)
 {
     int nb_frame;
 FILE* fp_in;
 FILE* fp_out;
 unsigned char serial[L_FRAME_COMPRESSED];
 short   synth[L_FRAME];
 int     bfi;
 va_g729a_init_decoder();
    fp_in = fopen("code.pcm", "rb");
    fp_out = fopen("rvoice.pcm", "wb");
 nb_frame = 0;
 while (fread(serial, sizeof(char), L_FRAME_COMPRESSED, fp_in) == L_FRAME_COMPRESSED)
 {
  bfi = 0;
  va_g729a_decoder(serial, synth, bfi);
  fwrite(synth, sizeof(short), L_FRAME, fp_out);
 }
 fclose(fp_out);
 fclose(fp_in);
 }
}
```

In the visual basic program I need to reference codec.dll in project manager .Then I can run the code and time its operation by:

☐ Collapse Copy Code

```
Dim sw As New Stopwatch
Dim mec As New masm.math
sw.Start()
mec.encoder()
Label2.Text = "encoder millisecs  " & sw.ElapsedMilliseconds
sw.Reset()
sw.Start()
mec.decoder()
Label4.Text = "decoder millisecs  " & sw.ElapsedMilliseconds
```

The times shown on the form are for 6 second buffer. If you use a 320(160 samples) byte buffer ie 20 msec the encode or decode takes less than 2 msec. This would be quite all right and well within the limits of 50% cpu time.

So far so good. Now take it to another computer. The first run on a vista machine gave the following problem:

Resolving Side-by-Side Configuration Issues

You may get this error with using a compiled c++ .exe or .dll

In XP:

The application has failed to start because the application configuration is incorrect. Reinstalling the application may fix this problem

OR in Vista:

The application has failed to start because its side-by-side configuration is incorrect. Please see the application event log for more detail.

A really good explaination of this error is available at:

http://buffered.io/2008/05/17/resolving-side-by-side-configuration-issues/

There are a number of problems commonly encountered with DLLs – especially after numerous applications have been installed and uninstalled on a system. The difficulties include conflicts between DLL versions, difficulty in obtaining required DLLs, and having many unnecessary DLL copies.

Windows XP introduced a solution called Side-by-Side Component Sharing, which loads separate copies of DLLs for each application that requires them (and thus allows applications that require conflicting DLLs to run simultaneously). This approach eliminates conflicts by allowing applications to load unique versions of a module into their address space, while preserving the primary benefit of sharing DLLs between applications (i.e. reducing memory use) by using memory mapping techniques to share common code between different processes that do still use the same module.

If you get this error then the machine does not have the required runtime dll in the windows\winSXS folder.
You can find which runtime .dll are required for the .exe or .dll by using dumpbin in VC++ express:

```
dumpbin /dependents g729e.exe
Dump of file g729e.exe
File Type: EXECUTABLE IMAGE
  Image has the following dependencies:
    MSVCR90D.dll
    KERNEL32.dll
```

The vista macine did not have MSVCR90D.dll

If you do dumpbin /all you will be able to see the manifest included in <xml> tages.

Manifests are XML files that accompany and describe side-by-side assemblies or isolated applications. Manifests uniquely identify the assembly through the assembly's <assemblyIdentity> element. They contain information used for binding and activation, such as COM classes, interfaces, and type libraries, that has traditionally been stored in the registry. Manifests also specify the files that make up the assembly and may include Windows classes if the assembly author wants them to be versioned. Side-by-side assemblies are not registered on the system, but are available to applications and other assemblies on the system that specify dependencies in manifest files.

A simple program to view the manifest in dll and exe files is available at:

http://weblogs.asp.net/kennykerr/archive/2009/01/02/manifest-view-support-for-dlls.aspx

For g729e.exe this gives:

dependant assembly:

assemblyIdentity type="win32" name="Microsoft.VC90.DebugCRT" version="9.0.21022.8" processorArchitecture="x86" publicKeyToken="1fc8b3b9a1e18e3b

The vista machine did not have this assembly in winSXS. The assembly is a folder containing the required .dlls in our case MSVCR90D.dll In xp you could just write the required assembly into winSXS however you cant in vista.

In Windows Vista, the directory WinSxS has much stronger protection on it than it did in Windows XP. The owner/group is now a SID named "Trusted Installer", a service SID used to start the TrustedInstaller service. Users other than the trusted installer are granted only generic-read/generic-execute by default. This increased protection ensures that only the trusted installer service is allowed to modify the servicing-related metadata and files.

I downloaded the latest :

The Microsoft Visual C++ 2008 Redistributable Package (x86) installs runtime components of Visual C++ Libraries required to run applications developed with Visual C++ on a computer that does not have Visual C++ 2008 installed.

This did not install the required dll into winSXS which by the way is 34 meg on the xp machine and 7.8gig!! on the vista machine.

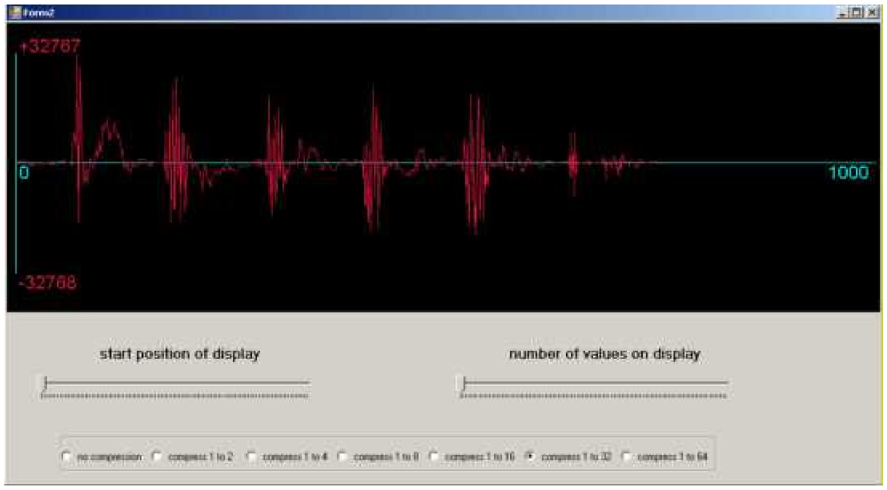
I installed the latest vc++ 2008 express on the vista machine and this finally did install the required runtime dll in winSXS.

If you compile the c++ on the same machine as you run it, or a program which uses it, you should not have this problem.

Debugging

I have included a simple pcm viewer which has helped me in my progress.

This shows a voice sample.



The solution file for all the programs I have dicussed are included in programs.zip

License

About the Author

carl morey

I identify with the starfish.
I may be really stupid and have to use visual basic but at least I'm happy.
Location: Australia

Member

Article Top ***Rate this article for us!*** *Poor* ○ ○ ○ ○ ○ *Excellent* Vote

IBM® Data Studio Developer
Free Visual Studio Training from AppDev
StateServer - powerful distributed caching

FAQ Search

Noise Tolerance Medium Layout Normal Per page 25 Update

New Message (Refresh)

-- There are no messages in this forum --

General News Question Answer Joke Rant Admin

BETA [Silver Members Only / Not Public]: Article page views by day.